FUNDAÇÃO GETÚLIO VARGAS

MBA em Gestão de Tecnologia da Informação

ALAN VICTORIA BAZAN

Utilização de Scrum no Desenvolvimento de Jogos Eletrônicos

São Paulo - SP

Maio de 2012

MBA em Gestão de Tecnologia da Informação

ALAN VICTORIA BAZAN

Utilização de Scrum no Desenvolvimento de Jogos Eletrônicos

Trabalho de Conclusão de Curso apresentado como requisito para colação de grau no Curso de Especialização em Gestão de Tecnologia da Informação da Fundação Getúlio Vargas, sob a orientação do Prof. Onófrio Notarnicola Filho.

São Paulo - SP

Maio de 2012

O Trabalho de Conclusão intitulado “Utilização de Scrum no Desenvolvimento de Jogos Eletrônicos” apresentada por Alan Victoria Bazan como requisito para colação de grau no curso de Especialização em Gestão de TI à Banca Examinadora da Fundação Getulio Vargas, obteve conceito _____ para aprovação.

BANCA EXAMINADORA

_____

Prof. Onófrio Notarnicola Filho

_____

_____

São Paulo, _____ de _____ de 20___

À minha família, pela infinita paciência e apoio.

AGRADECIMENTOS

Agradeço a meus familiares por estarem sempre presentes na minha vida e por me darem apoio nos momentos críticos.

À Fundação Salvador Arena pelas oportunidades oferecidas.

E aos amigos do grupo Forja Entertainment pelo ótimo trabalho e por ter cedido o Space Cowboy para o presente estudo.

RESUMO

O desenvolvimento de software é uma atividade muito complexa. Tecnologias mudam e evoluem rapidamente. A indústria de jogos eletrônicos partilha dessa complexidade, sendo ela aumentada por diversas outras áreas como arte visual, *design*, escrita, música e até psicologia. Para lidar com essa complexidade, custos e prazo presentes no projeto de jogo eletrônico os desenvolvedores devem seguir uma produção assertiva, em sintonia com os objetivos do projeto. E é isso que propõe esse trabalho: mostrar se a utilização de metodologias ágeis, exemplificada pelo framework Scrum, seria ideal para a área de desenvolvimento de jogos e quais os benefícios delas em relação a outras metodologias comumente utilizadas.

ABSTRACT

Software development is a very complex activity. Technologies change and evolve rapidly. The electronic game industry share this complexity, it is augmented by other areas such as visual art, design, writing, music and even psychology. To deal with this complexity, cost and time present in the design of video game, developers should follow an assertive production, aligned with the project objectives. And that's what this work proposes: to show if the use of agile methodologies, exemplified by the Scrum framework, would be ideal for the game development area and what their benefits in relation to other methodologies commonly used.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 PROBLEMA

As empresas de desenvolvimento de software buscam cada vez mais a melhoria do seu processo de desenvolvimento para evitar gastos excessivos, cancelamento de projetos e projetos falhos. A partir dessas necessidades, surgiram várias metodologias de desenvolvimento, entre elas as metodologias ágeis.

O Scrum é uma metodologia ágil que será usada como foco do estudo. Sabendo do motivo da sua criação frente às dificuldades de desenvolver um software, percebe-se que ela é bem semelhante às dificuldades de desenvolvimento de jogos.

Na área de jogos existem problemas com prazos, cancelamento de projetos e projetos falhos, assim como no desenvolvimento de software comum. Porém, esses itens possuem facetas específicas. Os problemas mais preocupantes de um jogo são: como perceber, o mais cedo possível, que a experiência do jogo será passada do modo que foi previsto e que essa experiência seja divertida? Como garantir que o software do jogo apresentará corretamente características tão subjetivas quanto diversão e experiência antes do projeto estar completo?

Solucionando esses problemas, é possível fazer com que se gaste menos tempo fazendo algo que está fora do esperado, percebendo um problema cedo e ajustando o projeto para uma linha correta.

1.2 CONTEXTUALIZAÇÃO

Uma das áreas mais lucrativas do momento é a área de entretenimento digital, principalmente a área de jogos. E uma das principais discussões decorrentes das pessoas envolvidas na área, principalmente as iniciantes e independentes, é: “qual o melhor jeito de se construir e controlar um projeto de jogo?”.

Muitos projetos de jogos, assim como projetos de software, são cancelados ou efetuados com falhas. Daí surge a preocupação de sempre ter melhores jeitos de se controlar, projetar e construir um projeto, da numerosa quantidade de falhas e do grande tempo, consequentemente dinheiro, desperdiçado em um projeto que pode fracassar.

A área de desenvolvimento de software sempre passou por esses problemas, e, a partir deles, foram desenvolvidas melhoras nas metodologias existentes e criadas novas

metodologias. Uma das últimas novidades foi a criação de metodologias ágeis e a sua adoção por grandes empresas de desenvolvimento de software.

Segundo o Agile Manifesto, um dos objetivos da metodologia ágil é responder rapidamente às necessidades do cliente e às mudanças de projeto com um software funcionando. Essa característica entra em sintonia com uma característica do projeto de um jogo, onde se deve garantir que a diversão e a experiência do jogo sejam relevantes para o jogador. Garantir que esses itens abstratos funcionem antes do projeto ser construído é um desafio quase que impossível. As metodologias ágeis podem entrar nessa área para ajudar no desafio de desenhar o jogo de maneira assertiva e garantir experiência e diversão o mais rápido possível, sendo capaz de moldar o projeto de acordo com as dificuldades e erros encontrados.

1.3 OBJETIVOS

Essa pesquisa tem como objetivo comparar as metodologias de desenvolvimento ágil com outras metodologias como a metodologia em cascata e espiral; e verificar como utilizar a metodologia ágil com o framework Scrum em projetos de software de jogos, levando em conta os pontos de sucesso de um jogo e o ambiente em que um jogo é desenvolvido, considerando que a equipe desenvolvedora de jogos é multidisciplinar, envolvendo programadores, artistas, designers, músicos, entre outros.

A motivação para esse tema veio de curiosidades de como construir efetivamente projetos de jogos seguindo os padrões subjetivos de *design* de jogos, que só são efetivamente comprovados na construção do projeto. Com as características principais das metodologias ágeis é possível que elas sejam a chave para obter sucesso no *design* e desenvolvimento de jogos.

1.4 PROCEDIMENTOS

A pesquisa será feita através de livros e artigos relacionados ao funcionamento e uso da metodologia Scrum para desenvolvimento de software, apresentando a história, motivos da criação, principais diretrizes e características, e quais as vantagens e desvantagens do seu uso.

Em seguida serão feitas pesquisas semelhantes relacionadas à área de jogos. Pesquisa de bibliografias a fim de apresentar a estrutura de uma empresa desenvolvedora de jogos,

tipos de profissionais, sua interdisciplinaridade, e quais os principais desafios da construção de um jogo de acordo com os princípios conhecidos de *design* de jogos.

A partir dessa pesquisa as duas áreas serão correlacionadas para observar onde o Scrum pode ser vantajoso e desvantajoso na área de desenvolvimento de jogos, e como utilizá-lo de forma efetiva em frente ao tipo de estrutura de uma organização voltada a esse ramo.

Será feito, também, um estudo de caso com um grupo independente de desenvolvimento de jogos: o grupo Forja. O grupo utilizará a metodologia Scrum em um projeto de jogo. O jogo a ser desenvolvido será o Space Cowboy: um jogo curto e de baixa complexidade para ser jogado na internet, voltado para o público casual.

Durante o desenvolvimento serão feitos relatos, para serem colocados na pesquisa, do andamento do projeto em cada etapa, das documentações geradas, das reuniões importantes (previstas pela própria metodologia), e do andamento do projeto. Serão feitas, também, análises do andamento do projeto, dos pontos positivos e negativos da metodologia.

1.5 ESTRUTURA DO TRABALHO

Esse trabalho está dividido em quatro capítulos com temas centrais relevantes para o estudo do framework Scrum no desenvolvimento de jogos eletrônicos.

O segundo capítulo apresenta um histórico e a situação atual da indústria de jogos eletrônicos (videogames) em um contexto mundial, relatando o mercado e descrevendo as características envolvidas na sua produção (profissionais envolvidos, aspectos de um jogo, entre outras).

No terceiro capítulo será apresentado o conceito de metodologia ágil, quais as suas características e motivos para a criação, além de se comparar com outras metodologias.

O quarto capítulo é apresentado o objeto central do estudo, o framework Scrum. Será mostrada a sua definição e características.

Por fim, será apresentado um estudo de caso: o desenvolvimento do jogo Space Cowboy, do grupo Forja Entertainment, que foi efetuado em cima do framework Scrum.

2 A INDÚSTRIA DE JOGOS ELETRÔNICOS

2.1 HISTÓRICO

Antes de se estudar o processo de desenvolvimento e metodologias utilizadas na produção de jogos eletrônicos, ou *games*, é preciso estudar o histórico dessa indústria, perceber o seu funcionamento, como encaminha a sua evolução e quais os erros cometidos no passado, e atualmente. Esse estudo poderá servir de catalisador para a descoberta de maneiras melhores de se construir *games* e, no caso desse trabalho, descobrir se as metodologias de desenvolvimento ágil e o framework Scrum (que será apresentado nos capítulos seguintes) se adéquam às necessidades e à realidade da área.

A indústria de *games* é bem recente. Diferente de muitas outras indústrias, ela nasceu no século XX, mais precisamente, na década de 50, durante a tensão da Guerra Fria (DISCOVERY CHANNEL, 2006).

Nesse período, os Estados Unidos e a União Soviética investiam muito em materiais bélicos e em tecnologias, incluindo supercomputadores para fazer cálculos e simulações para o uso de novas armas, mísseis e estratégias de batalha. Essas tecnologias evoluíram rapidamente, mostrando uma verdadeira corrida entre os dois ideais para mostrar quem estava mais preparado caso estourasse um conflito armado (DISCOVERY CHANNEL, 2006). O computador que se conhece no século XXI estava longe da realidade de seu uso nesse período. Eles eram conhecidos como instrumentos de auxílio à guerra.

Como mostrado no documentário A Era do Videogame (2006), as pessoas eram meros espectadores dessa "guerra", aguardando tensamente o momento em que um dos países apertasse o "botão vermelho" para iniciar o fim do mundo. As pessoas assistiam em suas TVs as possibilidades e a movimentação dos países em relação à guerra sem ter o controle da situação.

Enquanto o mundo se movia para uma possível guerra, em 1958 o físico William Higinbotham, que trabalhou na primeira bomba atômica, construiu um jogo chamado Tennis for Two (Figura 1), utilizando um osciloscópio. Esse jogo consistia em uma bolinha de luz que se movia de acordo com os botões apertados no osciloscópio (DISCOVERY CHANNEL, 2006).

Figura 1 - Recriação do Tennis For Two (BROOKHAVEN HISTORY, 2011).

Steve Slug Russell, programador da MIT (Massachusetts Institute of Technology) também entrou na área criando o jogo SpaceWar!, que representava os acontecimentos da época em meio à corrida espacial (RETROSPACE, 2009). As pessoas poderiam desligar as notícias que viam nas TVs e obter controle sobre a situação nos videogames (DISCOVERY CHANNEL, 2006).

Nessa época, o americano Nolan Bushnell resolveu investir na área de jogos. Ele foi o primeiro a tratar o videogame como um negócio. Abriu uma empresa chamada Atari e, junto com Al Alcorn, construiu o seu primeiro *arcade*: o Pong, lançado em 1972. Pong foi um grande sucesso, abrindo caminho à febre das pessoas pelos videogames (DISCOVERY CHANNEL, 2006).

Quando os jogos de videogames comerciais surgiram, a complexidade para a sua produção era imensa.

> In the beginning, video game development didn't require artists, designers, or even programmers. In the early seventies, games were dedicated boxes of components that were hardwired together by electrical engineers for a specific game. These games first showed up in arcades and later in home television consoles that played only one game […] (KEITH, 2010, p. 4).

Toda a produção de um jogo não estava contida apenas no software, a confecção dele era mais voltada à engenharia elétrica do que à engenharia de computação. Cada jogo tinha uma arquitetura de hardware única causando, para cada projeto, um reaproveitamento mínimo, ou quase inexistente dos jogos anteriores. Essa estrutura era usada na criação de vários *arcades* (Figura 2).

Figura 2 - Máquinas de jogos (*arcades*). (IGN)

> In the golden age of the video arcade, during the seventies and early eighties, games like Pac-Man, Asteroids, Space Invaders, and Defender were gold mines.[…] This new gold rush attracted quite a few prospectors. Many of these 'wanna-be' arcade game creators went bankrupt in their rush to release games. […] investment that was easily destroyed if the machines shipped with a poor game (KEITH, 2010, p. 5).

A construção de a*rcades* com hardware complexo não era uma jogada muito segura. A sua arquitetura era rígida, impedindo mudanças tanto na tecnologia quanto nas características dos jogos. Se um jogo não fizesse sucesso, todo o dinheiro gasto com a arquitetura, fabricação e distribuição das máquinas era jogado na lata do lixo, assim como os próprios *arcades*. Esse tipo de trabalho tinha que mudar.

Segundo Keith (2011), os fabricantes de jogos descobriram que o uso de microprocessadores permitia uma maneira melhor de se construir jogos. Com esses microprocessadores, eles poderiam construir o jogo em uma única estrutura de hardware e mudar apenas o conteúdo programado (software), permitindo criar jogos diversos sem precisar construir uma estrutura física específica. A Atari acatou essa ideia e desenvolveu o console doméstico Atari 2600 (DISCOVERY CHANNEL, 2006). "The specific logic of each game moved from hardware to software. With this change, the game developers turned to programmers to implement games" (KEITH, 2010, p. 4).

Keith (2011) cita que desenvolver o software de um jogo era menos custoso do que desenvolver um hardware de jogo e representava apenas um pedaço pequeno do custo total da

produção. Dessa maneira os desenvolvedores poderiam descartar jogos ruins e desenvolver outro ou modificar até que esteja bom antes de fabricar o hardware dedicado para o jogo. “This iterative approach helped fuel the release of consistently high-quality games from companies like Atari.” (KEITH, 2010, p. 6)

Nos anos 80 a indústria de videogames, tanto para *arcades* quanto para consoles, estava repleta de empresas desenvolvedoras de jogos. Todos tinham a ideia de que qualquer jogo daria um retorno grande em dinheiro. Dessa maneira, muitos jogos foram lançados às pressas, sem planejamento e sem qualidade alguma, gerando a decadência da indústria americana de jogos (DISCOVERY CHANNEL, 2006).

> The market decline in the mid-eighties was caused by the increase proportion of inferior games released because of falling hardware costs. […] home consoles allowed almost anyone to create and mass-produce games cheaply. […] When the market became flooded with poor-quality games, consumers spent their money elsewhere (KEITH, 2010, p. 6).

Enquanto o mercado de games quebrava no ocidente, outros países entraram nesse ramo com sucesso. Na Russia, em 1985, Alexey Pajitnov criava o jogo Tetris, que incluía uma nova perspectiva para os jogos: o uso do raciocínio. E no Japão, a indústria se instalou e evoluiu, produzindo jogos com qualidade e com um bom *design*, abordando características como personagens e narrativas, destacando-se títulos como Pac-Man (Namco, 1980), seguido por Donkey Kong (Nintendo, 1981) e Super Mario Bros. (Nintendo, 1984), salvando o mercado de videogames domésticos (DISCOVERY CHANNEL, 2006).

Foram criados jogos de raciocínio e investigação para os computadores pessoais, visando o público adulto. Dentre esses jogos destacam-se o Mystery House (Sierra On-Line, 1970) e King's Quest (Sierra On-Line, 1984) (Figura 3), destacando-se pela trama e pela qualidade gráfica (DISCOVERY CHANNEL, 2006).

Figura 3 - Mistery House (Sierra On-Line, 1970) e King's Quest (Sierra On-Line, 1991).

Os acontecimentos do fim e após a guerra fria, no final da década de 80 e a década de 90, influenciou diretamente a indústria de jogos. Não só os desenvolvedores de videogames amadureceram, mas os jogadores também. Eles queriam coisas novas, experiências novas. Assim empresas como a Sega e a Nintendo se preocupavam para lançarem jogos de sucessos, como Sonic (Sega, 1991), e tecnologias melhores, como o uso de CDs, iniciado como um projeto em parceria da Nintendo com a Sony, que, devido a desavenças contratuais, foi quebrada, levando a Sony a investir sozinha na tecnologia lançando o console Playstation (Sony, 1994), levando a indústria de jogos em um patamar elevado, com alta qualidade gráfica e sonora (DISCOVERY CHANNEL, 2006).

Com isso a indústria de jogos evoluiu incrivelmente, lançando cada vez mais jogos e cada vez mais novas tecnologias, sendo uma indústria altamente lucrativa, como mostrada na Figura 4.

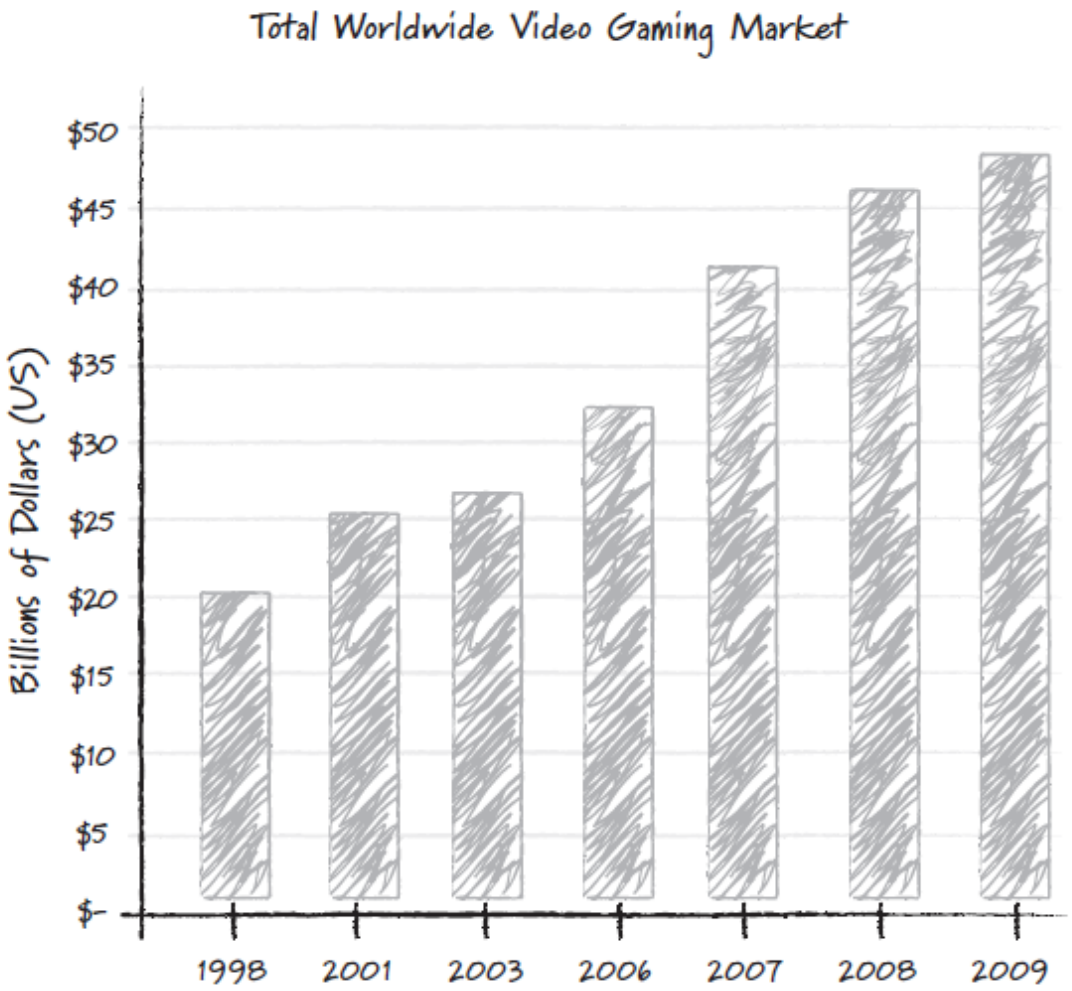

Figura 4 - Crescimento do mercado de videogames (KEITH, 2010, p. 8).

2.2 METODOLOGIAS

Com o grande avanço das tecnologias a construção de um jogo foi se tornando cada vez mais complexo e caro. “A lone programmer could no longer write a game that leveraged the full power of evolving consoles. […] Software and art production became the greater part of the cost of releasing a game to market.” (KEITH, 2010, p. 6)

O risco de se construir um jogo ruim causava um resultado desastroso. Muitos profissionais estavam envolvidos em sua construção: programadores, artistas, *designers*, músicos, entre outros e os projetos estavam cada vez maiores e mais complexos. Então,

> To reduce the increasing risk, many companies adopted waterfall-style methodologies used by other industries.[…] The waterfall methodology employed the idea of developing a large software project through a series of

> phases. Each phase led to a subsequent phase more expensive than the previous (KEITH, 2010, p. 6)

O modelo *waterfall* (ou cascata) é uma metodologia para desenvolvimento de software cujo processo era dividido em etapas, sendo que o produto passa por uma progressão linear desde a concepção, passando pelos requerimentos, *design*, codificação e teste (LAPLANTE & NEILL, 2004).

> Introduced (but not named) by Winston Royce[4] in 1970 when computer systems were monolithic, number-crunching entities with rudimentary front ends (by today's standards) and users' needs were filtered through the partisan minds of the computer illuminati building the systems.(LAPLANTE & NEILL, 2004)

Nessa época as pessoas que faziam os requisitos dos *softwares* eram os próprios programadores, com uma pequena participação dos *stakeholders*. Como os usuários não participam do desenvolvimento, raramente os requisitos se alteravam. Nesse cenário, a metodologia *waterfall* funcionava com sucesso. Porém, hoje em dia, a participação do usuário no desenvolvimento é crucial. Se o sistema não supre as suas necessidades, eles o rejeitarão (LAPLANTE & NEILL, 2004).

O modelo em cascata consiste na seguinte divisão: a fase inicial consiste no planejamento de como o software será desenvolvido e quais as suas características; na fase central o software é desenvolvido; e na fase final seus componentes são integrados e o software é testado. Cada fase tem o propósito de reduzir o risco do desenvolvimento antes de se passar para a próxima fase (KEITH, 2010, p. 6). A Figura 5 apresenta as fases do modelo em cascata comumente usado para a realidade do desenvolvimento de jogos.

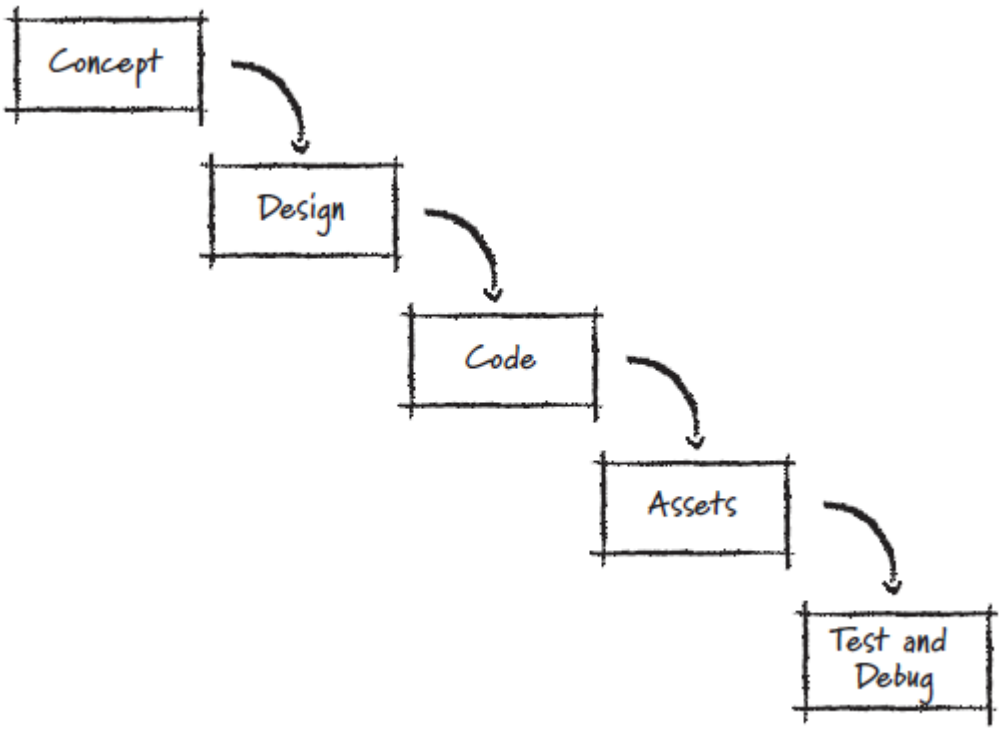

Figura 5 - Waterfall game development (KEITH, 2010, p. 7)

O maior problema do modelo em cascata, segundo Schwaber (1995), é a sua natureza linear. O processo não define como responder a um resultado inesperado em qualquer uma das fases. Com essa abordagem, o desenvolvimento do software não responde a mudanças de necessidades, esperando sempre um processo rígido de desenvolvimento.

Segundo Boehm (1988), outro problema do modelo em cascata é a ênfase em documentações muito elaboradas. Dá-se mais valor em manter as especificações da documentação do que a sua interação com o usuário. Em aplicações como sistemas operacionais e compiladores essa é a metodologia mais efetiva, mas não em outros tipos de aplicações, principalmente com interação do usuário final. “Document-driven Standards have pushed many projects to write elaborate specifications of poorly understood user interfaces and decision-support functions, followed by the design and development of large quantities of unusable code.” (BOEHM, 1988)

Para resolver os problemas da metodologia em cascata foi definido um novo modelo de desenvolvimento: o modelo espiral (Figura 6). Esse modelo apresenta ciclos iterativos de desenvolvimento.

> The model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its levels of elaboration, from an overall concept of operation document down to the coding of each individual program. (BOEHM, 1988)

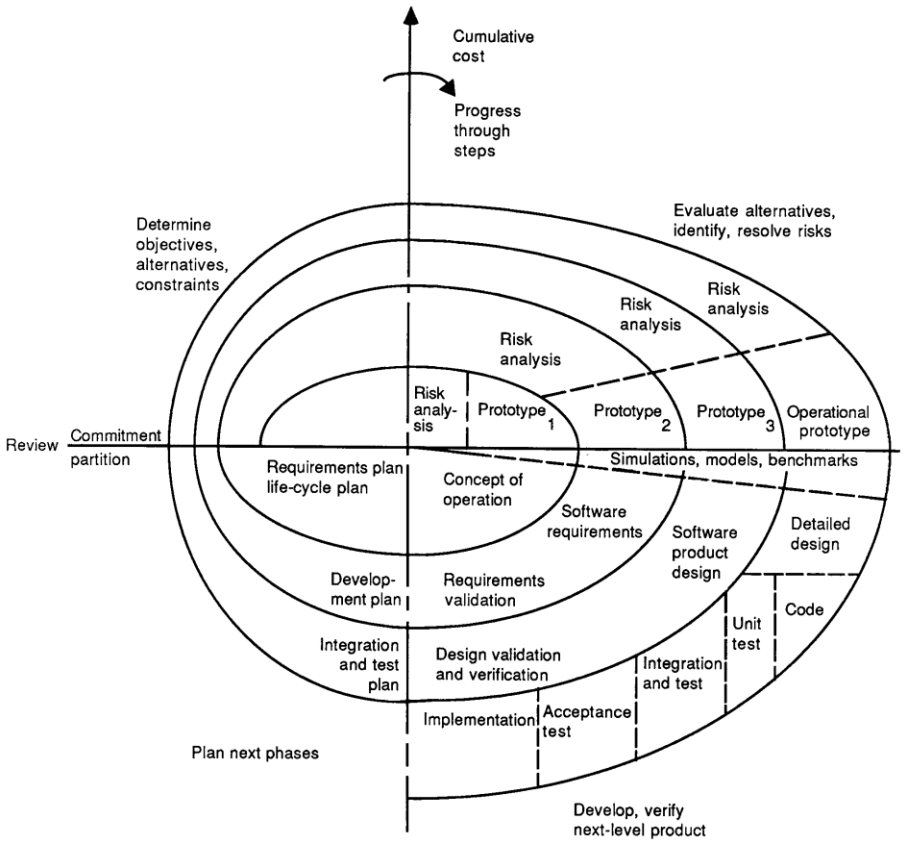

Figura 6 - Metodologia Espiral. (BOEHM, 1988)

Segundo Boehm (1988), o modelo em espiral avança ciclicamente entre quatro fases:

- Identificação dos objetivos, alternativas e recursos (pessoas, custos, prazo, etc.);
- Avaliação de riscos e formulação de estratégias para suas resoluções, que podem envolver protótipos, simulações, benchmarking, questionários, modelos analíticos, entre outros;
- Verificação do resultado dos protótipos em relação ao seu risco e desenvolvimento do projeto evoluindo como no modelo cascata (analise, *design*, construção e teste);
- Revisão do que foi desenvolvido até agora e das condições da organização quanto ao projeto.

A metodologia espiral resolveu muito dos problemas presentes na metodologia em cascata, porém, outros problemas continuam presentes. Segundo Schwaber (1995), essa metodologia progride através das camadas de desenvolvimento. O resultado dos protótipos diz como continuar o desenvolvimento, mas o processo ainda continua linear. "Requirements work is still performed in the requirements phase, design work in the design phase, and so forth, with each of the phases consisting of linear, explicitly defined processes." (SCHWABER, 1995, p. 5)

2.3 EQUIPE

2.3.1 Game Designers

Game designers são, segundo Salen e Zimmerman (2003), pessoas que projetam a jogabilidade, as regras e as estruturas de um jogo. Ele é responsavel por elaborar o *design* do jogo.

"Game design is the act of deciding what a game should be.[…] To decide what a game is, you must make hundreds, usually thousands of decisions." (SCHELL, 2008, p. XXIV)

O *game designer* possui a visão e o objetivo do projeto. Seu objetivo é garantir que o jogador viva uma determinada experiência de forma significativa através do jogo, tomando decisões sobre a mecânica, estética, tecnologia e história (SCHELL, 2008) e garantir, também, que todos os envolvidos sigam o mesmo objetivo ao longo do caos do desenvolvimento (BATES, 2004). Como o *design* do jogo está tão ligado com a forma com que ele é programado, visualizado, e apoiado pela música e voz, o *game designer* deve sempre estar apoiando e guiando todos os componentes da equipe (FULLERTON, 2008).

2.3.2 Programadores

Na área de jogos, os programadores têm um papel fundamental, já que um jogo eletrônico, em sua essência, é um software. Todos os envolvidos na tecnologia, desde codificadores até engenheiros de redes e sistemas, estão relacionados com a programação (FULLERTON, 2008).

Segundo Bethke (2003), a área de tecnologia na indústria de jogos é muito diversificada. Ela está presente nas áreas padrões de desenvolvimento de software e se

estendem para áreas menos comuns, como a artística, surgindo profissionais como programadores 3D, programadores de efeitos visuais e gráficos. Outros tipos de programadores também podem ser encontrados nessa área, como programadores de inteligência artificial, arquitetos de redes (para projetos de jogos multijogadores), programador de interfaces e de áudio, que controlam os efeitos sonoros no jogo, incluindo efeitos de áudio tridimensionais.

As pessoas relacionadas à área de tecnologia têm como objetivo construir protótipos, o motor do jogo e ferramentas que auxiliem a produção, de acordo com as necessidades das outras equipes (design, arte, etc.) (FULLERTON, 2008).

2.3.3 Artistas

Os artistas são todos aqueles que produzem desenhos, artes, modelos 3D, animações e qualquer outro objeto visual inerente ao jogo. Os artistas estão presentes em todos os aspectos visuais de um jogo: desde a interface até os efeitos especiais (BATES, 2004).

Segundo Bethke (2003), costumava-se ter apenas um artista responsável pela confecção do mundo do jogo e dos personagens, mas com a grande participação da arte em um projeto e o aumento da complexidade dos jogos atuais, a criação de arte se tornou complexa. Muitas empresas têm o seu próprio departamento artístico, pois consideram uma ótima arte uma vantagem competitiva (BATES, 2004).

Com a grande diversidade dos tipos de arte existentes em um jogo, vários estilos de artistas são requeridos, incluindo desenhistas de personagens (exemplificado na Figura 7), ilustradores, animadores, desenhistas de interface e artistas 3D (exemplificado na Figura 8) (FULLERTON, 2008 & BETHKE, 2003).

Figura 7 - Planejamento de Aida, personagem de Unreal 2 (Epic Games).

Figura 8 - Animação 3D no 3DStudioMax (BATES, 2004).

2.3.4 Outras Mídias

Na indústria de jogos ainda existem infinitos tipos de mídias que podem ser utilizadas. Segundo Fullerton (2008), o jogo pode exigir a participação de escritores, músicos, engenheiros de som, tecnologia de captura de movimento (Figura 9), instrutores de karatê ou dubladores. Esses profissionais geralmente não fazem parte da equipe e são contratados por um curto período de tempo.

Figura 9 - Captura de movimento (BATES, 2004).

Raramente a equipe que desenvolve um jogo possui todas as habilidades necessárias para a produção e tende a recorrer a terceiros, principalmente ao que diz respeito aos efeitos sonoros, músicas e vozes, que são as características mais presentes nos jogos atualmente (BATES, 2004).

2.4 DESIGN DE JOGOS

Nessa sessão serão mostradas as principais características do *design* de jogos e suas principais problemáticas, para assim fazer um estudo sobre a aderência de uma metodologia ágil com a realidade do desenvolvimento de jogos.

O *game design*, ou *design* de jogos, como explicado anteriormente, trata-se da tomada de decisão sobre o jogo, envolvendo diversas características, visando fazer com que o jogador vivencie uma experiência significativa.

> When people play games, they have an experience. It is this experience that the designer cares about. Without the experience, the game is worthless. […]Everything we've ever seen (look at that sunset!), done (have you ever flown a plane?), thought (why is the sky blue?), or felt (this snow is so cold!) has been an experience (SCHELL, 2008, p.10)

A Figura 10 apresenta um mapa de *design* de jogos proposto por SCHELL (2008). Esse mapa apresenta os pontos com que o *designer* deve se preocupar para fazer o *design* do jogo.

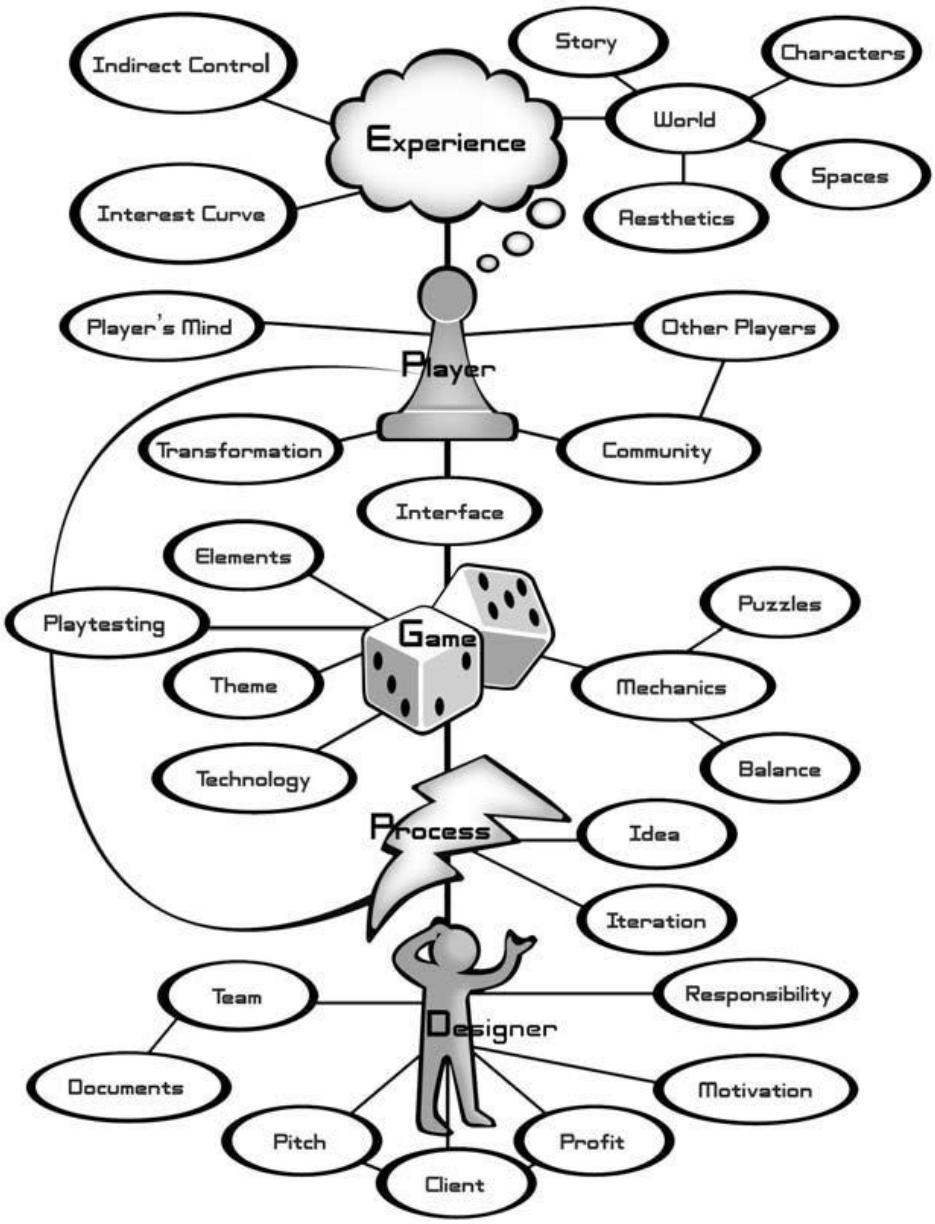

Figura 10 - Mapa de *Game Design* proposto por SCHELL (2008)

É possível perceber através desse mapa que o desenvolvimento de jogos é complexo e que muitos parâmetros devem ser levados em conta para que seja possível atingir o sucesso

de um projeto. A construção de um jogo envolve um processo criativo, construção de ideias, construção e balanceamento da mecânica de jogo para não torná-lo injusto ou frustrante para o jogador, criação de quebra-cabeças e desafios, construção de uma interface com o jogador para que a sua experiência seja fluída, construção do ambiente e do mundo do jogo em relação às suas regras, história e estética, além da preocupação com a equipe e com as tecnologias envolvidas.

2.4.1 Problemas

2.4.1.1 Experiência do Jogador e o Playtest

Um dos problemas mais comuns na área de jogos é conseguir fazer com que o jogo ofereça realmente uma experiência significativa ao jogador.

> A good game designer should always be thinking of the player, and should be an advocate for the player. Skilled designers […] [think] about the player, the experience of the game, and the mechanics of the game all at the same time. Thinking about the player is useful, but even more useful is watching them play your game. The more you observe them playing, the more easily you'll be able to predict what they are going to enjoy. (SCHELL, 2008, p.106)

A partir dessa afirmação, como descobrir se o jogo atingiu seu objetivo antes que o seu público tenha contato com ele? E como perceber que o jogador realmente terá a experiência que o *designer* espera?

> "[…] experiences are what a game designer creates. These experiences can only happen in one place — the human brain. Entertaining the human brain is hard because it is so complex — it is the most complex object in the known universe. […] Even worse, most of its workings are hidden from us." (SCHELL, 2008, p.114)

O *designer* deve conseguir balancear o jogo para que o jogador tenha surpresas sem fim, que a experiência se mantenha significativa por todo o jogo. E a única maneira de atingir esse estado é jogar o jogo e alterá-lo continuamente até que esse estado apareça (SCHELL, 2008).

É comum ver que o *designer* tome a sua imaginação como verdade. Imaginando que suas decisões são ideais para que o jogo faça sucesso. Isso pode fazer com que a equipe trabalhe em uma ideia incerta, causando um possível fracasso do projeto. Por isso é necessário efetuar *playtestings*, testes em que se joga o jogo a fim de perceber se a experiência do jogo está condizente com o planejado, se o jogo é divertido, ou para capturar qualquer outro erro de *design*. Os processos de *playtest* devem ser efetuados o quanto antes para que os problemas

sejam identificados antes, evitando desenvolvimento desalinhado com o objetivo e verificando se o time está desenvolvendo o jogo certo para o público certo (SCHELL, 2008).

"The whole point of playtesting is to make clear to you that some of the decisions you were completely comfortable with are completely wrong. You need to find these things out as soon as possible, while there is still time to do something about them." (SCHELL, 2008, p. 391)

2.4.1.2 Feature Creep

Feature Creep é um termo utilizado quando os *stakeholders* requisitam que novas funcionalidades ou características sejam adicionadas ao produto depois que o seu escopo foi definido. Isso pode acontecer devido ao surgimento de novas necessidades ou caso as características originais não alcancem as expectativas (KEITH, 2010).

"unless you are making your game as a hobby, or you are independently wealthy, the client is probably paying you to make the game. And if they don't like the way things are going, it's literally game over." (SCHELL, 2008, p. 416). Então, é certamente esperado que aconteçam mudanças no decorrer do projeto. Os desenvolvedores devem estar preparados a esse tipo de ocorrência, e preparados para evitar o cancelamento dos projetos, estando prontos a atender às mudanças.

Segundo Keith (2010), o *feature creep* e as mudanças são inevitáveis e nem sempre são ruins. O problema é quando essas mudanças são feitas sem a alteração do cronograma e do orçamento quando mais trabalho é adicionado. E muitas desenvolvedoras aceitam esse tipo de mudança para evitar que o projeto seja cancelado.

> Opportunities to add value to the game are identified throughout the project, but with a tight schedule and workload, they either have to be ignored or have to be added at great peril to the deadline. Unfortunately, swapping out planned features for new ones that require the same amount of effort is not an option. Feature creep tends to expand the total scope. (KEITH, 2010, p. 17)

Os desenvolvedores nem sempre sabem tudo sobre o jogo. O conhecimento e a percepção do que funciona e do que não funciona só acontece quando o ele é jogado. A maneira ideal para evitar surpresas desastrosas no escopo do projeto é trabalhar com versões jogáveis desde o seu início (KEITH, 2010). A Figura 11 mostra o nível de incerteza sobre os objetivos do jogo de acordo com as etapas do projeto.

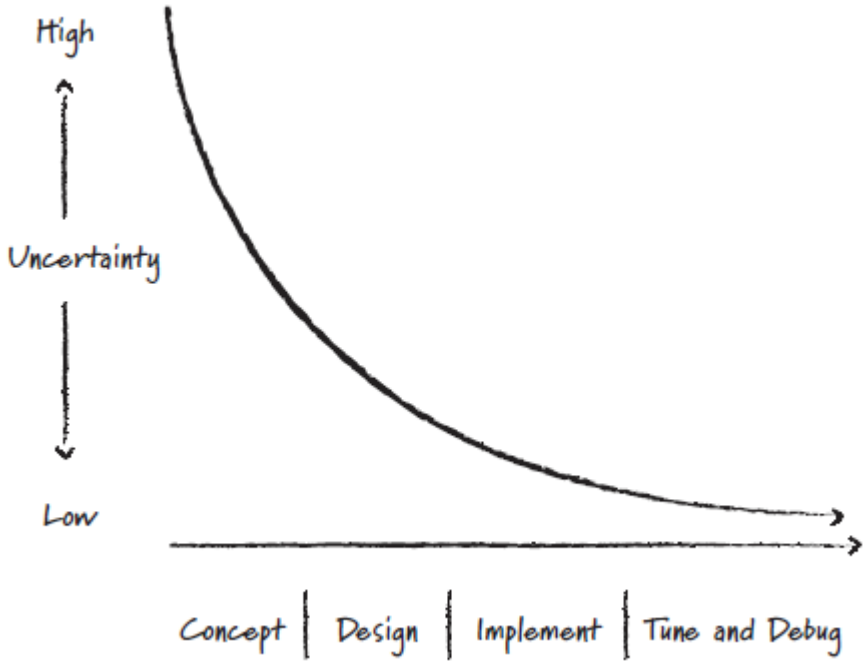

Figura 11 - Incerteza durante o projeto de jogo. (KEITH, 2010, p. 18)

Para lidar com esse tipo de problemática, a desenvolvedora, junto com os *game designers,* devem estar cientes da necessidade da alteração e como ela deve ocorrer. Muitos clientes sabem onde querem chegar mas não sabem exatamente como, podendo exigir uma característica que pode, ao invés de melhorar, acabar com o sucesso do produto. "A big part of the designer's job is to help the client figure out what they want. This is just like listening to your audience — you must get to know the client better than he knows himself." (SCHELL, 2008, p. 418)

2.4.1.3 Cronograma

Todo projeto, principalmente quando envolve cliente e, consequentemente, dinheiro precisa ter um cronograma. No caso de projetos de jogos, a equipe desenvolvedora precisa estimar, de acordo com o escopo do projeto, o tempo necessário para o seu término. Porém, vários problemas podem acontecer nesse processo.

> Task estimation is not an exact science. Even when we estimate simple things in daily life, such as running an errand at the store, unanticipated problems crop up and throw off our estimates. Traffic will jam, or the lines at the store will be long. The accuracy of estimates drops when more complex tasks are estimated, such as those for making games. (KEITH, 2010, p. 18)

Para se estimar o tempo para as tarefas, muitas variáveis devem ser consideradas: a diferença de experiência e produtividade entre diferentes pessoas; a quantidade de trabalho que determinada pessoa possui, em paralelo, em determinado período; a estabilidade do software e das ferramentas utilizadas; e a incerteza quanto ao nível de retrabalho necessário para polir determinada tarefa até que ela se mostre significativa e divertida para o jogador (KEITH, 2010).

2.4.1.4 Produção Desafiadora

A produção de jogos é altamente desafiadora. Sua natureza multidisciplinar torna o desenvolvimento e a comunicação entre as áreas complexa. Os jogos contam com grandes equipes, envolvendo programadores, artistas, *designers*, músicos, entre outros mais. As consequências, caso o projeto não tenha sucesso, são desastrosas. Segundo Keith (2010) o desafio da produção, frente à realidade da indústria, é maximizar a eficiência, minimizar a perda de tempo e criar previsibilidade.

> Predictability is more important during production. Production represents a great deal of work. Dozens of characters and levels have to be built before a game is shipped. Production often accompanies a major staffing increase or engagement of an outsource company. Mass-producing assets such as characters and levels shouldn't start early. The game mechanics and asset budgets must be established to create proper assets on the first pass to avoid expensive rework. (KEITH, 2010, p. 19)

Keith (2010) mostra que, quando a equipe entra muito rápido na produção ela ainda não sabe realmente quais os seus objetivos e quais as diretrizes para o desenvolvimento. Se os artistas começam a construir personagens e cenários, e os programadores, mecânicas, motores e ferramentas, e o escopo muda de acordo com a incerteza, muito trabalho pode ser jogado fora e terá que ser construído novamente. Mas se a equipe demora a entrar na fase de produção (principalmente pela busca da diversão e da experiência) a equipe é forçada a entrar na produção pelo cronograma e pelo cliente.

3 DESENVOLVIMENTO ÁGIL

Mesmo com a evolução das metodologias de desenvolvimento, passando do modelo cascata para o espiral, elas ainda se prendem ao conceito de definir as entregas no início do projeto. O desenvolvimento ágil é um avanço da metodologia espiral, trabalhando em cima do

conceito de iterações, ou ciclos de desenvolvimento (SCHWABER, 1995). Cada iteração é tratada como o desenvolvimento um projeto único, onde seu objetivo é entregar uma gama de funcionalidades ou subsistemas (SCHWABER, 1995).

> Each iteration consists of all of the standard Waterfall phases, but each iteration only addresses one set of parsed functionality. The overall project deliverable has been partitioned into prioritized subsystems, each with clean interfaces. Using this approach, one can test the feasibility of a subsystem and technology in the initial iterations. Further iterations can add resources to the project while ramping up the speed of delivery. This approach improves cost control, ensures delivery of systems (albeit subsystems), and improves overall flexibility. (SCHWABER, 1995, p. 6)

Segundo Keith (2010), projetos de TI fracassam com uma frequência crescente com o uso de metodologias como a em cascata. Isso fez com que surgissem muitos livros e artigos que definiam melhores praticas para o desenvolvimento de software. Algumas dessas práticas e metodologias promoviam o desenvolvimento iterativo, onde cada iteração deveria conter uma fatia do projeto incluindo todas as fases do desenvolvimento (requisitos, design, produção, teste e implementação). "The iterations could be as short as a week but included analysis, design, coding, integration, and testing within that time frame rather than spreading each of them out over years as they could be on a waterfall project" (KEITH, 2010, p. 13).

Muitas metodologias surgiram seguindo esse padrão, até que, em 2001, um grupo de pessoas experientes na área resolveu identificá-las como metodologias ágeis. A partir disso, eles criaram o "manifesto ágil", um documento que engloba os princípios e os valores dessas metodologias (KEITH, 2010).

De acordo com o Agile Manifesto, os princípios do manifesto ágil são:

- Satisfazer o cliente através de entrega contínua e adiantada;
- Tirar vantagem nas mudanças de requisitos para oferecer vantagem competitiva ao cliente;
- Entregar frequentemente software funcionando em curtos espaços de tempo;
- Pessoas do negócio e do desenvolvimento devem estar diariamente juntas para o desenvolvimento do projeto;
- Motivação dos indivíduos oferecendo ambiente e suporte que eles precisarem, e confiança no desenvolvimento do trabalho;
- Método mais eficiente e eficaz de transmitir informação é na conversa face a face;
- Software funcionando é a base do progresso;
- Processos ágeis promovem desenvolvimento em ritmo constante;

- Excelência técnica e de *design* aumenta agilidade;
- Simplicidade é fundamental;
- Melhores d*esigns* provêm de equipes auto-organizáveis;
- Reflexão regular de como se tornar mais eficaz e ajustar comportamento de acordo.

O Agile Manifesto, além dos princípios citados acima, direciona o uso das metodologias ágeis para valorizar mais:

- Indivíduos e interação entre eles do que processos e ferramentas;
- Software em funcionamento do que documentação abrangente;
- Colaboração com o cliente do que negociação de contratos;
- Responder às mudanças do que seguir um plano.

3.1 VALORES

3.1.1 Indivíduos e interações mais que processos e ferramentas

Cada vez mais se usam processos e ferramentas para controlar projetos que estão cada vez maiores. Em um projeto de jogo participam cerca de 100 pessoas que trabalharão por alguns anos antes do mesmo ficar pronto. Essa quantidade de pessoas e o longo prazo levaram à criação de hierarquias gerenciais, cronogramas e documentos de *design*, que tentam prever todos os requerimentos para o jogo ser divertido. Todas essas características são julgadas necessárias frente às características do ambiente (KEITH, 2010).

Keith (2010) aborda que a indústria de jogos é uma mescla de diversas disciplinas, e que é importante que essas disciplinas se interajam entre si, suprindo as necessidades e fraquezas de cada área de uma maneira colaborativa e efetiva. “For example, it is important for an animator who discovers a bug in the animation technology to work with an animation programmer as quickly as possible” (KEITH, 2010, p.25). Mas com a extensa quantidade e preocupação com processos, hierarquias e ferramentas, esse auxílio entre as área pode sofrer um atraso. Ao invés do programador ter autonomia de ajudar o animador, ele pode estar dependente da autorização de seu líder, causando uma cadeia de comunicação (Figura 12).

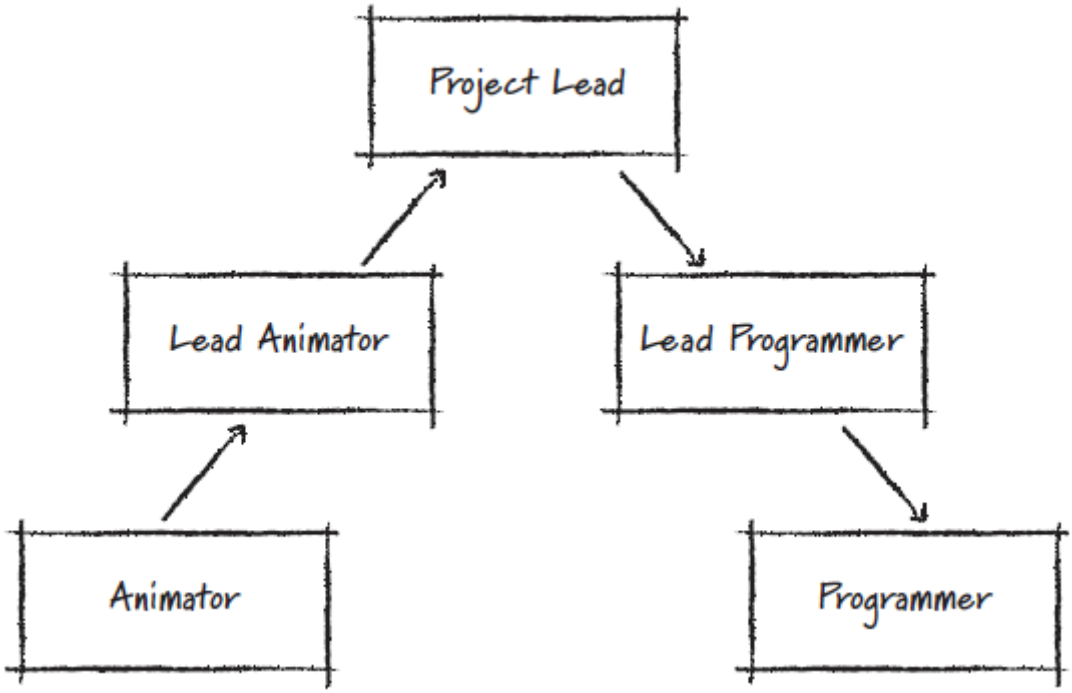

Figura 12 - Exemplo de cadeia de comunicação. (KEITH, 2010, p. 25)

"So, what is happening in the big picture?

- More than 100 people from various disciplines on one team
- Thousands of unpredictable problems that can introduce wasted time and effort
- Inflexible plans and tools to manage people who can't predict and quickly react to these problems
- Hierarchies of management that can lead to further waste" (KEITH, 2010, p. 26)

As metodologias ágeis diferem das tradicionais nessa questão. Ao invés da equipe estar rigidamente dependente de liderança, é promovido que ela seja independente, solucionando muitos dos problemas por conta própria. A equipe fica responsável por gerenciar problemas menores enquanto a liderança mantém o foco em características mais abrangentes (KEITH, 2010).

Essa característica, segundo Keith (2010) faz com que o time comece a se preocupar, cada vez mais, com problemas maiores. Quando a equipe percebe que consegue lidar com problemas menores, ela ganha confiança para resolver problemas de maior complexidade e importância. Isso faz com que eles se preocupem em:

- Criar melhores estruturas para a equipe para resolver mais problemas reduzindo dependências externas e aumentando o foco em sua resolução;
- Identificar riscos rapidamente e reportá-los antes que se transformem em problemas;
- Identificar e melhorar a própria capacidade de liderança.

3.1.2 Software em funcionamento mais que documentação abrangente

Para um projeto de jogos, é necessário que exista algum tipo de documentação do seu *design*. Isso se torna necessário para que as publicadoras, detentoras de licenças e os outros *stakeholders* tenham uma noção clara sobre os objetivos do projeto, para prever uma direção do projeto para tomadas de decisão, incluindo decisões como investir ou não no projeto. Esse tipo de documentação comunica aos envolvidos, inclusive na equipe de desenvolvimento, as características do produto, seus objetivos, e diretrizes (KEITH, 2010).

Uma ideia um pouco contrária é incentivada pelas metodologias ágeis. A construção de documentos é necessária, porém, uma documentação abrangente e detalhada pode atrapalhar. Muito tempo pode ser gasto para se definir características e documentá-las, sendo que o sucesso e a necessidade dela ainda são duvidosos.

> […] detailed plans […] can create work that is not necessary. If all the assumptions about the weapon system were implemented before discovering what was fun about it, much of that work is wasted. If the project sticks to the detailed plan, then it won't be the best game possible. (KEITH, 2010, p. 27)

3.1.3 Colaboração com o cliente mais que negociação de contratos

Uma característica constantemente presente na indústria de jogos é a existência de contratos definindo uma serie de *milestones* (marcos). Cada *milestone* propõe uma data de entrega de determinadas funcionalidades do jogo. E muitas vezes o pagamento aos desenvolvedores é feito a partir do sucesso de um *milestone*. Muitos desenvolvedores, principalmente os independentes, necessitam desse pagamento. Isso faz com que sejam desenvolvidas as características presentes no contrato e com que os desenvolvedores se preocupem em terminar a *milestone* na data correta. Assim, muitas características percebidas com o decorrer do projeto que poderiam melhorar o jogo são descartadas com o receio de que quanto mais características forem adicionadas, mais trabalho será adicionado, e a data de entrega (e do pagamento) serão adiados (KEITH, 2010).

> Many developers who miss a milestone payment miss payroll; that is a very bad thing for them to do. The contract is an impediment to change. […] On the other side, a publisher doesn't have the full freedom to add features or change the milestone definition when they think the game would benefit from the change. The contract impedes working with the developer to fix the game. (KEITH, 2010, p. 27)

Segundo Keith (2010), uma relação entre desenvolvedores e publicadores baseada em contratos fixos pode levar a uma perda mutua. Os desenvolvedores e publicadores são barrados a adicionar mudanças ou novas características por não abrir mão de características descritas no contrato, construindo uma relação não amigável. Nas metodologias ágeis é essencial que ambas as partes envolvidas no projeto tenham um nível de confiança. Colaboração entre desenvolvedores e publicadores devem ser mais valiosos do que contratos fixos. Porém, poucas publicadoras permitem o desenvolvimento de um produto sem uma documentação detalhada. Muitos contratos em um ambiente ágil, fora dessa área, possuem o padrão de pagamento por tempo e material. O cliente paga por custo de tempo de trabalho e de materiais utilizados no fim de cada iteração. Nesse ambiente o cliente deve ter confiança no desenvolvedor, esperando que ele tenha sabedoria para gastar o capital investido; e o desenvolvedor deve ter confiança que o cliente não cancelará o projeto ou o investimento sem boas razões (KEITH, 2010).

3.1.4 Responder a mudanças mais que seguir um plano

As metodologias ágeis tem como base, segundo Keith (2010) as seguintes características: planejar o projeto em cima do que é conhecido; e, testar e descobrir o que não é conhecido através das iterações. Muitas das características de um projeto são colocadas à prova durante o seu desenvolvimento, e pode se descobrir que essas características não estão de acordo com o objetivo do projeto ou não trazem experiências significativas para o jogador (no caso de um jogo), sendo necessária a mudança no escopo do projeto por causa de uma definição incerta.

> Expanding project teams and ever-increasing feature and hardware complexities have driven managers to turn to increasingly detailed planning. […] defined processes are best applied when we have certainty about the technology required by a project and well-understood requirements we know will develop into a hit game. These two criteria are rarely seen. Not only do our platforms change frequently, but creating a fun, innovative game is always challenging. (KEITH, 2010, p. 28)

Schwaber (1995) mostra que metodologias como cascata e espiral partem de uma definição de escopo construído no início do projeto. E metodologias iterativas partem de um

plano abrangente de características e datas de entrega, desvendando as reais características e prazos durante o projeto, baseado no ambiente corrente a partir de um método empírico, trazendo fases de análise, design e desenvolvimento para cada uma das iterações.

O tipo de abordagem presente nos modelos em cascata e espiral também causam problemas na indústria de jogos.

> Projects with more than 100 developers, with costs exceeding tens of millions of dollars to develop, are now common. Many of these projects go over budget and/or fail to stay on schedule. […] The rising cost of game development and the impending death of the hit-or-miss model has created a crisis for game development in three main areas: less innovation, less game value, and a deteriorating work environment for developers. (KEITH, 2010, p. 10)

Keith (2010) apresenta que é padrão entre essas empresas evitar falhas evitando correr riscos. Com essa abordagem as desenvolvedoras e publicadoras se prendem a ideias repetidas, cujo fator de sucesso é historicamente garantido. Com isso, as empresas estão ficando cada vez menos inovadoras. Muitos dos jogos desenvolvidos atualmente são sequências de jogos que já deram certo, ou jogos baseado em licenças (como filmes) com o intuito de se prender a sucessos anteriores.

3.2 POR QUE SER ÁGIL?

A indústria de software está rumando para as metodologias ágeis. Mas qual o motivo dessa migração? Quais vantagens essa metodologia traz para os desenvolvedores e para os clientes?

Segundo Keith (2010), o mercado, inclusive a de videogames, está empurrando os desenvolvedores para oferecer produtos de maior qualidade e menor custo. As características das metodologias ágeis fazem com que projetos sejam desenvolvidos sem desperdício de trabalho e com foco contínuo na realidade atual do cliente. Nas metodologias anteriores, como a metodologia em cascata, são feitas previsões sobre como será o produto e seu comportamento, mas essas visões não são inteiramente claras. Na produção de um jogo, a equipe aprende com o seu desenvolvimento.

> As we develop a game, we are learning. We learn what plays well with the controller, what looks good on the target platform, and how to make the game run fast enough with enough artificial intelligence (AI) characters to make it challenging. We create knowledge every day. (KEITH, 2010, p. 21)

Keith (2010) mostra que todo esse conhecimento é impossível de ser inteiramente previsto na concepção de um plano de projeto. As incertezas são reduzidas no decorrer do

tempo e do desenvolvimento do produto. “Agile development focuses on building knowledge about value, cost, and schedule and adjusting the plan to match reality.” (KEITH, 2010, p. 21)

3.3 CUSTOS

O primeiro problema que abre caminhos para a utilização de metodologias ágeis é o custo. Projetos de software e de jogos envolvem milhões de dólares para serem produzidos. Seguindo a lógica de Keith (2010), um jogo de sucesso para PC é vendido a cerca de 60 dólares. Se esse jogo vende meio milhão de cópias, o dinheiro arrecadado nas vendas será de 30 milhões de dólares. Desse dinheiro, três quartos vão para custo de licenças, distribuição, marketing e publicidade, restando 7,5 milhões de dólares para a desenvolvedora. Muitos projetos de jogos custam mais do que isso, e a maioria deles não chega a atingir meio milhão de unidades, fazendo com que muitos jogos falhem e empresas sejam quebradas.

Schell (2008) mostra uma situação semelhante. A Figura 13 mostra a rota do dinheiro baseado na venda de um jogo. De uma venda de um jogo de 50 dólares, ela é diluída entre vários envolvidos: publicadora, agências de publicidade, entre outros, ficando apenas 8 dólares de cada venda para o desenvolvedor.

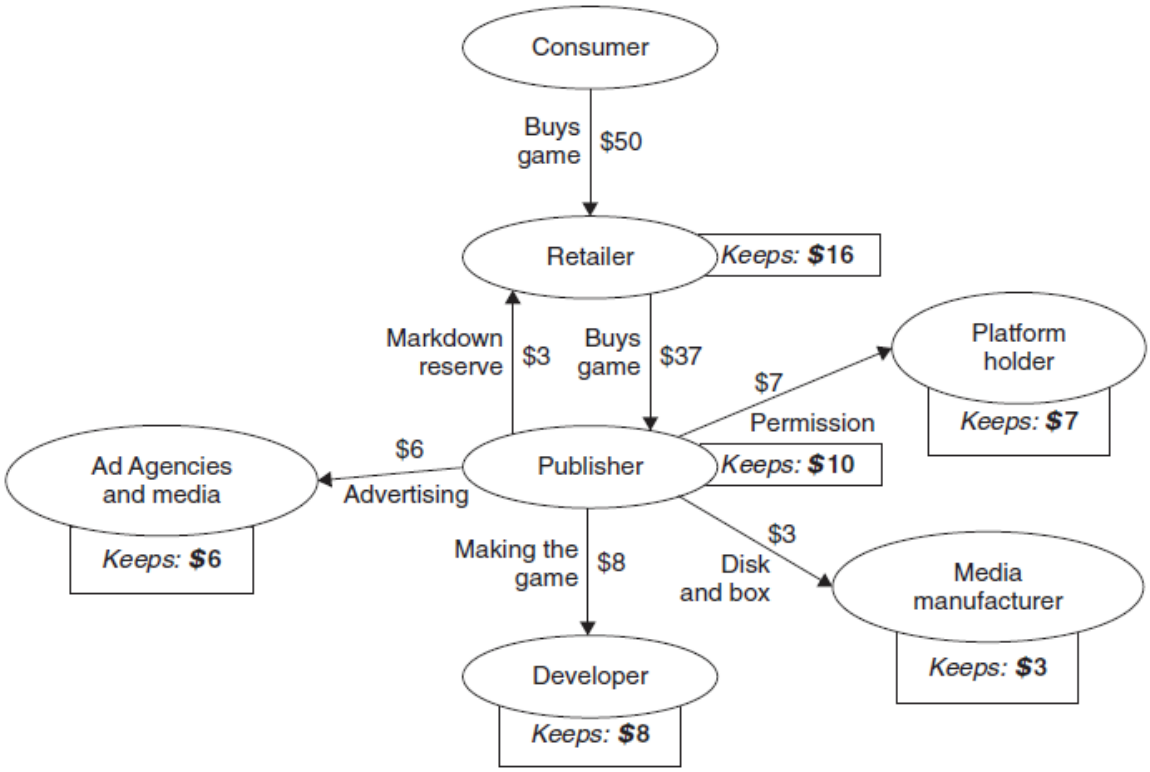

Figura 13 - Mapa do dinheiro envolvido em um jogo. (SCHELL, 2008, p. 435)

Para diminuir os custos do projeto, as publicadoras e desenvolvedoras acatam menos riscos e estão tomando, segundo Keith (2010), as seguintes providências:

- Procurando oportunidades de terceirizar criação de conteúdos e codificação;
- Apostando em soluções *middleware* (subsistemas para auxílio no desenvolvimento);
- Reduzindo a quantidade de conteúdo para o jogo (vendendo um jogo com oito horas de jogo ao invés de dezesseis).

Publicadoras reduzem o número de jogos falhos fazendo o seguinte:

- Apostando em propriedades licenciadas como jogos baseados em filmes;
- Apostando em sequências e franquias antigas que fizeram sucesso no passado;
- Oferecendo poucas oportunidades para novas ideias, reduzindo a qualidade dos jogos no mercado.

3.4 DIVERSÃO EM PRIMEIRO LUGAR

Um dos benefícios do desenvolvimento iterativo proposto nas metodologias ágeis é o desenvolvimento do produto em passos pequenos. Ao invés de se gastar tempo e dinheiro em um plano grande, são adicionadas, de forma incremental, as características que satisfazem o cliente da forma mais econômica (KEITH, 2010).

Keith (2010) afirma que o cliente, para um jogo, é a pessoa que compra e joga. Quanto mais o jogo for divertido e significativo para o jogador, mais vendas acontecerão. Isso faz com que a busca da diversão seja o ponto principal do desenvolvimento de um jogo. E isso só é possível descobrir da mesma maneira que o seu cliente: jogando. É possível prever certas características para um jogo, mas não se pode garantir que ele seja divertido ou significativo durante o planejamento sem mesmo tê-lo experimentado. Esse é o problema com projetos produzidos em metodologias em cascata. Eles mostram um progresso mínimo nos primeiros dois terços do projeto, como mostrado na Figura 14.

> By ‘finding the fun’ first, the project team finds the value early in the project rather than trying to retrofit it at the end. […] Making changes at the end of a production cycle that affects every production asset is a lot more expensive than discovering the change before most of the assets are created. (KEITH, 2010, p. 24)

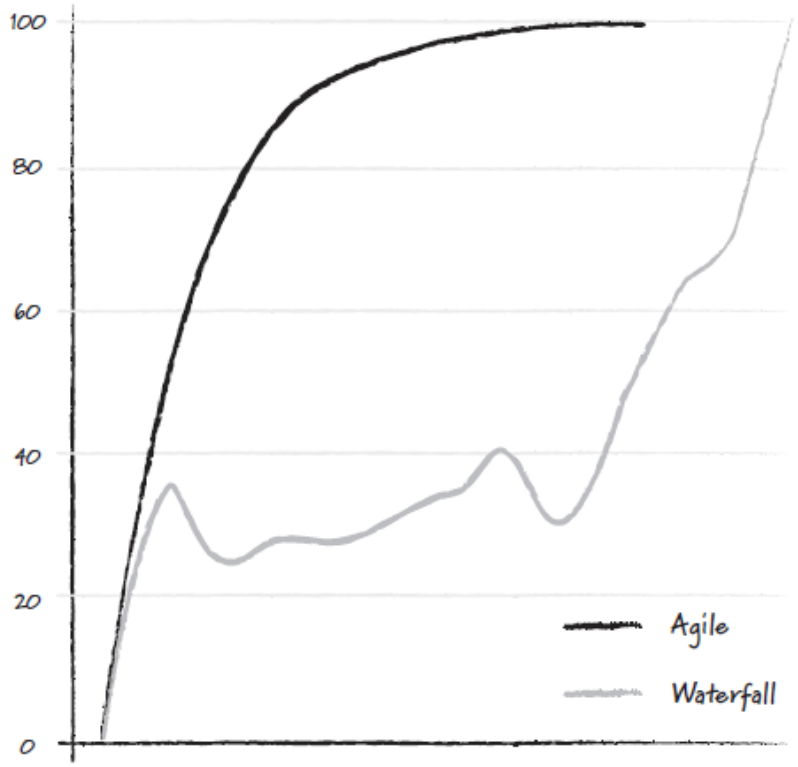

Figura 14 - Comparação do encontro da diversão pelo tempo entre o método cascata e ágil. (KEITH, 2010, p. 23)

Os projetos ágeis levam o desenvolvimento para uma abordagem com foco, primeiramente, no valor. Isso acontece quando o projeto é iterativo, ao trazer as características para um estado final ou próximo disso em ordem de prioridade de acordo com o seu valor. A publicadora é beneficiada nessa abordagem, pois ela espera que o valor seja apresentado o quanto antes. Assim os *stakeholders* não perdem anos de custo e esforço em projetos que não serão divertidos. Encontrar a diversão do jogo primeiro força a equipe a deixar o jogo melhor, questionando-o a cada passo (KEITH, 2010). “Iteration enables the project team to easily measure the cost of development and improve the efficiency of how groups of people work together. It creates a culture of continual improvement that can reduce the cost of developing games.” (KEITH, 2010, p. 24)

3.5 DESENVOLVIMENTO ITERATIVO

Um projeto ágil é composto de uma série de iterações de desenvolvimento. Iterações são espaços curtos de tempo de trabalho que duram de duas semanas a um mês. O progresso de qualquer projeto ágil se encontra no acontecimento dessas iterações.

A equipe de desenvolvimento, segundo Keith (2010), implementa as características que tem valor para o cliente (chamadas de *user stories*, ou estórias do usuário). Durante cada iteração, em um projeto de jogo, todos os elementos presentes no desenvolvimento são incluídos, incluindo conceituação, *design*, codificação, criação de conteúdos (artes, sons, etc.), teste e polimento.

No final de cada iteração, segundo Keith (2010), o produto é questionado e revisado, resultando em um novo ponto de partida e influenciando os objetivos das próximas iterações.

> This is an example of using the 'inspect and adapt' principle. […] Teams and costumers inspect the progress of a game every iteration and adapt the plan to address what is valuable and what is not. Teams inspect how they are working together every iteration and adapt their practices to improve their effectiveness. (KEITH, 2010, p. 29)

Como mostrado por Keith (2010), projetos ágeis não evitam planejamentos. Eles apenas permitem que o plano não seja seguido à risca do início ao fim do projeto, eles permitem a alteração dos planos de acordo com o desenvolvimento. Em projetos em Cascata, as datas de entrega guia o projeto para o objetivo planejado. Quando o projeto chega a seu objetivo, os envolvidos percebem que poderiam ter alcançado um objetivo melhor. Porém, nesse momento o projeto costuma não ter mais tempo e dinheiro disponível (Figura 15).

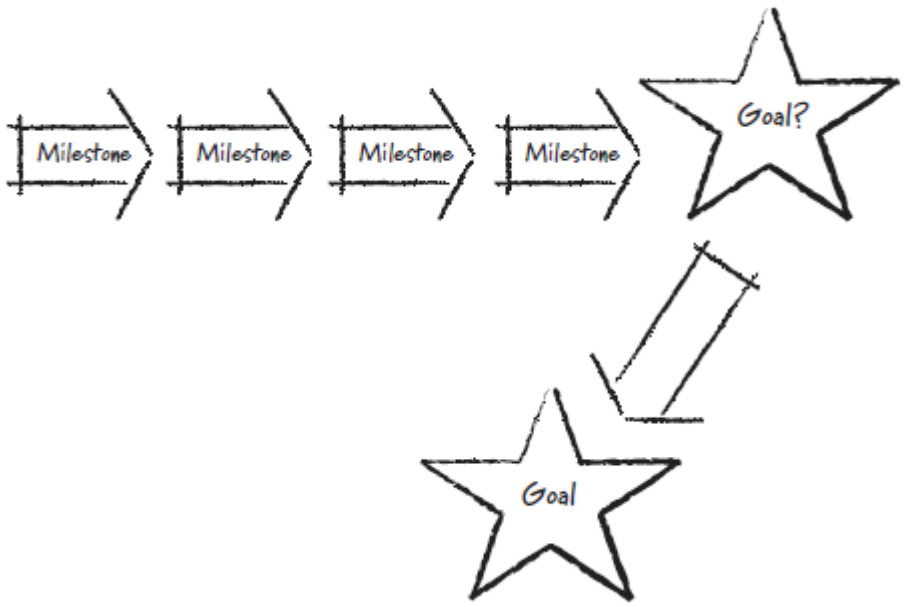

Figura 15 - Milestones steps toward a goal. (KEITH, 2010, p. 30)

Projetos ágeis também rumam para um objetivo, mas ele é ajustado à partir do desenvolvimento iterativo e da característica de se revisar e adaptar o projeto a cada ciclo. Ao invés de se seguir um plano à risca, o projeto é adaptado durante o desenvolvimento, desviando-se para um objetivo mais desejável (Figura 16) (KEITH, 2010).

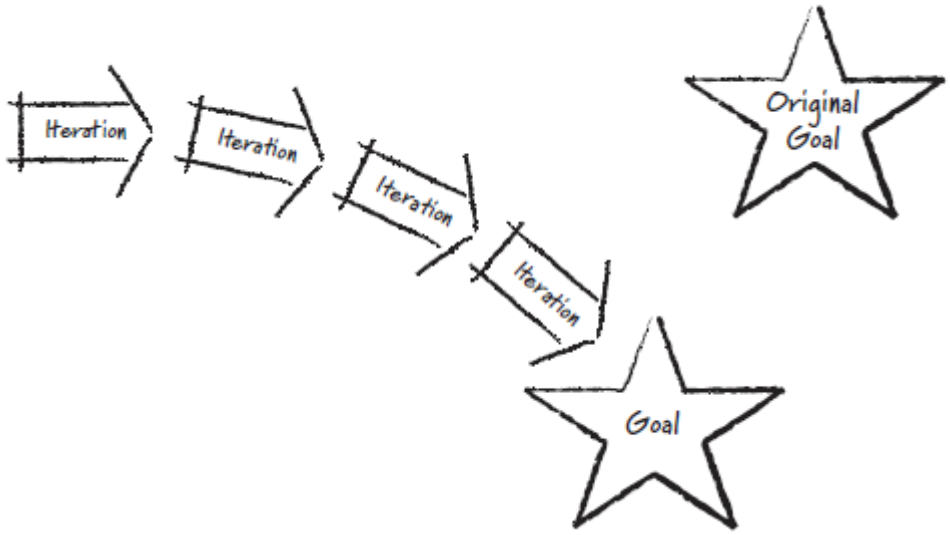

Figura 16 - Iterations toward a goal. (KEITH, 2010, p. 30)

Dessa maneira, a característica iterativa das metodologias ágeis faz com que o projeto seja desenvolvido enquanto a equipe aprende as reais necessidades do cliente e as melhores maneiras de se atingir seus objetivos. O projeto é ajustado e guiado para um caminho melhor do que planejado no início do projeto.

4 FRAMEWORK SCRUM

O Scrum é um *framework* baseado nas metodologias ágeis. Com ele, segundo Schwaber e Sutherland (2011), as pessoas podem desenvolver problemas complexos e adaptáveis, desenvolvendo sua solução sem abrir mão da produtividade e da criatividade, permitindo a entrega de produtos com o maior valor possível.

Ao contrário das metodologias e *frameworks* existentes, o Scrum não é um processo ou uma técnica para o desenvolvimento de produtos. Ele é um *framework* que permite empregar diversos processos ou técnicas existentes. O Scrum clarifica a eficácia do gerenciamento do produto e das praticas de desenvolvimento, abrindo caminho para a equipe melhorá-las (SCHWABER & SUTHERLAND, 2011).

O Scrum e seus entregáveis são baseados em uma série de variáveis. Essas informações são utilizadas para se construir um plano inicial para o desenvolvimento do software, porém, essas variáveis mudam constantemente durante o desenvolvimento. Essas variáveis, como mostrado por Schwaber (1995), são:

- Requerimentos do consumidor – quanto o sistema atual precisa de melhoria;
- Pressão de tempo – prezo necessário para se conseguir vantagem competitiva;
- Concorrência – o que a concorrência está disposta a fazer e o que é preciso para superar;
- Qualidade – a qualidade requerida frente às variáveis anteriores;
- Visão – que mudanças são requeridas nesse estágio para alcançar a visão da do sistema;
- Recursos – que pessoas e investimentos estão disponíveis.

Essas variáveis mudam durante o desenvolvimento e a sua revisão é importante para que o projeto atinja à sua melhor meta. “A successful development methodology must take these variables and their evolutionary nature into account.” (SCHWABER, 1995, p. 3)

Existem outras variáveis no desenvolvimento de software, como as variáveis de ambiente. Softwares são desenvolvidos em um ambiente altamente complicado. As variáveis de ambiente, segundo Schwaber (1995), incluem:

- Disponibilidade de profissionais habilidosos – quanto mais nova é a tecnologia, ferramentas, métodos e domínios, menor é a gama de profissionais especializados;

- Estabilidade da tecnologia de implementação – quanto mais nova a tecnologia maior a necessidade de balanceá-la com outras tecnologias ou procedimentos manuais;
- Estabilidade e poder das ferramentas - quanto mais nova e mais ponderosa a ferramenta de desenvolvimento, menor a gama de profissionais qualificados e maior a instabilidade da ferramenta;
- Efetividade dos métodos – quais os métodos de modelagem, teste, controle de versão, e design e quão efetivos e eficientes provaram ser;
- Experiência no domínio – se profissionais habilidosos estão disponíveis nos vários domínios, incluindo negócios e tecnologia;
- Novas características – que características completamente novas serão adicionadas, e em que grau ela vai se encaixar nas funcionalidades correntes;
- Metodologia – se a abordagem de desenvolvimento de sistemas e o uso dos métodos selecionados promovem flexibilidade ou se é rígido e detalhado;
- Concorrência – o que a concorrência fará durante o projeto e que novas funcionalidades serão anunciadas ou entregues;
- Tempo/Investimento – quanto tempo e investimento estão disponíveis no início do projeto e no decorrer do desenvolvimento;
- Outras variáveis – outros fatores que devem ser respondidos durante o projeto para garantir o sucesso do software.

A partir dessas variáveis a equipe pode manter o controle do desenvolvimento e ajustá-lo de acordo com as necessidades, entregando incrementos e produtos com um real valor para o cliente.

4.1 A EQUIPE DO SCRUM

4.1.1 Product Owner

O *Product Owner* (dono do produto), como o próprio nome diz, é a pessoa responsável pelo produto. A sua tarefa é fazer com que o produto tenha o maior valor possível, e tirar o máximo proveito da equipe de desenvolvimento.

> The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals. (SCHWABER & SUTHERLAND, 2011, p.5)

Esse profissional é responsável, também, por gerenciar o *Product Backlog*, artefato do Scrum que define as características do projeto, que será descrito com detalhes mais à frente. Schwaber & Suitherland (2011) mostram que, além das tarefas descritas, o dono do produto também é responsável por:

- Expressar claramente os itens do *Product Backlog*;
- Ordenar e priorizar os itens do *Product Backlog* para melhor alcançar os objetivos;
- Garantir o valor do trabalho feito pela equipe de desenvolvimento;
- Garantir que o *Product Backlog* esteja visível e claro para todos, e que as próximas tarefas também sejam visíveis;
- Garantir que o time de desenvolvimento entenda os itens do *Product Backlog* no nível necessário.

4.1.2 Equipe de Desenvolvimento

A equipe de desenvolvimento no Scrum tem um papel diferente comparado a outras metodologias. Ao invés da organização gerenciar a equipe, no Scrum ela é incentivada a organizar o seu próprio trabalho. A sinergia resultante desse método aperfeiçoa a eficiência e eficácia da equipe de desenvolvimento (SCHWABER & SUTHERLAND, 2011).

Schwaber & Sutherland (2011) mostram que no SCRUM as equipes de desenvolvimento possuem as seguintes características:

- Elas são auto-organizáveis. A própria equipe define como transformar o *Product Backlog* em incrementos de potenciais funcionalidades;
- São multifuncionais. Possuem todas as habilidades necessárias para produzir um incremento do produto;
- O Scrum não intitula os membros da equipe, apenas os definem como desenvolvedores. A funcionalidade de um determinado membro é definida de acordo com o trabalho que ele fará. Não há nenhuma expectativa em manter um membro da equipe em determinada função;
- Membros individuais da equipe podem ter habilidades especializadas e áreas de foco, mas essas habilidades são contadas como habilidades presentes na equipe como um todo;
- E as equipes não contém subequipes responsáveis por áreas específicas como análise de negócios e teste.

Segundo SCHWABER & SUTHERLAND (2011), a equipe de desenvolvimento deve ser pequena o suficiente para ser ágil e grande o suficiente para produzir um trabalho significativo.

> Fewer than three Development Team members decreases interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment. Having more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to manage. The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog. (SCHWABER & SUTHERLAND, 2011, p.6)

4.1.3 Scrum Master

O *Scrum Master* é o responsável por garantir que a metodologia (no caso o próprio Scrum) seja entendida e aplicada. Ele deve garantir que a equipe adira à teoria, praticas e regras do Scrum, servindo como um líder e servidor para a equipe (SCHWABER & SUTHERLAND, 2011)

Segundo Schwaber & Sutherland (2011), o *Scrum Master* serve as pessoas envolvidas no Scrum. Ele serve o dono do produto nos seguintes aspectos:

- Encontrando técnicas efetivas para o gerenciamento do *Product Backlog*;
- Comunicar claramente a visão e os objetivos do *Product Backlog* à equipe de desenvolvimento;
- Ajudando e ensinando a equipe de desenvolvimento a criar itens claros e concisos para o *Product Backlog;*
- Entendendo o planejamento e objetivos em longo prazo em um ambiente empírico;
- Entendendo e praticando a agilidade;
- E facilitando os eventos do Scrum quando requisitados ou necessários.

E serve a equipe de desenvolvimento nos seguintes pontos:

- Instruindo a equipe de desenvolvimento na auto-organização e no cruzamento das funções e habilidades presentes na equipe;
- Ensinando e liderando a equipe de desenvolvimento para criar produtos de alto valor;
- Removendo impedimentos para o progresso da equipe de desenvolvimento;
- Facilitando os eventos do Scrum quando requisitados ou necessários;

- E instruindo a equipe de desenvolvimento em organizações onde o Scrum não é inteiramente entendido e adotado.

4.2 ARTEFATOS

4.2.1 Product Backlog

Schwaber e Sutherland (2011) apresentam o *Product Backlog* como uma lista de tudo que será necessário conter no produto a ser desenvolvido. Todas as mudanças na construção do produto deverão partir de requerimentos descritos nesse documento. Os conteúdos, disponibilidade e ordenação dos itens são de responsabilidade do dono do produto.

O *Product Backlog* lista todas as características, funções, requerimentos, melhorias e consertos, constituindo em uma lista de mudanças a serem feitas no produto em um próximo lançamento ou versão. Os itens presentes nessa lista devem conter sua descrição, prioridade e estimativa para desenvolvimento (SCHWABER & SUTHERLAND, 2011).

O *Product Backlog* nunca está completo. No início do projeto ele contém apenas ideias gerais e requerimentos melhor conhecidos. À medida que o projeto e o ambiente em que ele será utilizado evoluem, o *Product Backlog* também evolui. Ele muda constantemente para gerar um produto com o maior valor possível, identificando as necessidades do cliente para obter um produto apropriado, competitivo e útil (SCHWABER & SUTHERLAND, 2011).

> As a product is used and gains value, and the marketplace provides feedback, the Product Backlog becomes a larger and more exhaustive list. Requirements never stop changing, so a Product Backlog is a living artifact. Changes in business requirements, market conditions, or technology may cause changes in the Product Backlog. (SCHWABER & SUTHERLAND, 2011, p.13)

No início do projeto os objetivos do projeto ainda não estão totalmente certos e definidos. Quanto mais se evolui no desenvolvimento, mais claros os objetivos e necessidades vão se tornando. O *Product Backlog* inicia com ideias gerais e, durante o desenvolvimento, o dono do produto e a equipe de desenvolvimento colaboram para rever e revisar os seus itens, acarretando em adição de níveis cada vez mais detalhados de acordo com a evolução do projeto (SCHWABER & SUTHERLAND, 2011).

4.2.2 Sprint Backlog

O *Product Backlog* contém as características do produto, já o *Sprint Backlog*, segundo Schwaber e Sutherland (2011), contém os itens do *Product Backlog* que serão desenvolvidos no *Sprint* (iteração) e o plano para desenvolver e entregar o incremento do produto. ”The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality.” (SCHWABER & SUTHERLAND, 2011, p.14)

O *Sprint Backlog* define qual o trabalho que a equipe de desenvolvimento irá fazer para transformar itens do *Product Backlog* em um incremento do produto. Ele deixa visível todo o trabalho que deverá ser feito para que a equipe alcance o objetivo do *Sprint* (SCHWABER & SUTHERLAND, 2011).

Assim como o as características do produto, o *Sprint Backlog* é mutável durante o decorrer do desenvolvimento. A equipe o modifica à medida que ela aprende mais sobre as necessidades para alcançar o objetivo do *Sprint*, começando com aspectos gerais e conhecidos e completando-o no decorrer do *Sprint*.

4.3 EVENTOS

4.3.1 Sprint

Sprint, segundo Schwaber e Sutherland (2011), é um período de tempo de um mês ou menos no qual será desenvolvida uma série de funcionalidades, ou subsistema, potenciais para o projeto. O objetivo do *Sprint* é desenvolver um incremento para suprir as necessidades mais conhecidas ou mais prioritárias do produto final. *Sprints* tem durações curtas e consistentes, e quando um termina, o próximo se inicia imediatamente (SCHWABER & SUTHERLAND, 2011)

Segundo Schwaber e Sutherland (2011), durante o Sprint:

- Nenhuma alteração que poder afetar ou mudar o objetivo do *Sprint* serão feitas;
- A composição da equipe e seus objetivos de qualidade continuam constantes;
- E o escopo do projeto pode ficar mais claro sendo possível renegociá-lo com o dono do produto e a equipe de desenvolvimento.

Os *Sprints* duram, no máximo, um mês. Se o período para se desenvolver certas funcionalidades é muito extenso, os objetivos e necessidades - junto com o escopo – podem mudar e a complexidade e o risco podem crescer (SCHWABER & SUTHERLAND, 2011). “Sprints enable predictability by ensuring inspection and adaptation of progress toward a goal at least every calendar month. Sprints also limit risk to one calendar month of cost.” (SCHWABER & SUTHERLAND, 2011, p.8)

Segundo Schwaber e Sutherland (2011), o *Sprint* pode ser cancelado caso o seu objetivo se torne obsoleto. Isso pode ocorrer pelo avanço da tecnologia, mudança nas condições do mercado, entre outros motivos. Cancelamentos devem ser feitos quando o produto do *Sprint* não é mais necessário, porém, pela curta duração da iteração, cancelamentos costumam ser raros.

4.3.2 Sprint Planning Meeting

O *Sprint Planning Meeting*, Reunião de Planejamento do *Sprint*, é uma reunião com tempo máximo de oito horas para *Sprints* de um mês, sendo esse período diminuído proporcionalmente para *Sprints* de menor duração, cujo objetivo é planejar as diretivas e o trabalho que será realizado durante a iteração. Esse plano é criado através da colaboração de toda a equipe do Scrum (SCHWABER & SUTHERLAND, 2011).

Segundo Schwaber e Sutherland (2011), essa reunião possui duas partes (divididas em tempos iguais). Cada parte é direcionada para responder as seguintes questões:

- O que será entregue como incremento do produto no próximo *Sprint*?
- E como o trabalho necessário para entregá-lo será feito?

Na primeira parte a equipe de desenvolvimento prevê as funcionalidades que serão desenvolvidas para o próximo *Sprint*. “The Product Owner presents ordered Product Backlog items to the Development Team and the entire Scrum Team collaborates on understanding the work of the Sprint.”(SCHWABER & SUTHERLAND, 2011, p.9)

Para efetuar essa parte do planejamento é necessário ter o *Product Backlog*, o último incremento (resultado) do produto, conhecimento da capacidade da equipe e do seu desempenho nos *Sprints* anteriores. A equipe de desenvolvimento decide quantos itens do *Product Backlog* serão desenvolvidos com a orientação do dono do produto. Apenas a equipe de desenvolvimento pode estimar o que é possível de se desenvolver durante o *Sprint* (SCHWABER & SUTHERLAND, 2011).

Depois que a equipe prevê os itens que serão desenvolvidos, a equipe define um objetivo para o *Sprint* (*Sprint Goal*). “The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment” (SCHWABER & SUTHERLAND, 2011, p.9) .

Na segunda parte a equipe de desenvolvimento planeja como o objetivo do *Sprint* será alcançado e como transformar as funcionalidades em um incremento pronto do produto. Os itens selecionados do *Product Backlog* junto ao plano do desenvolvimento formam o que é chamado de *Sprint Backlog* (SCHWABER & SUTHERLAND, 2011, p.9).

> The Development Team usually starts by designing the system and the work needed to convert the Product Backlog into a working product increment. Work may be of varying size, or estimated effort. However, enough work is planned during the Sprint Planning meeting for the Development Team to forecast what it believes it can do in the upcoming Sprint. Work planned for the first days of the Sprint by the Development Team is decomposed to units of one day or less by the end of this meeting. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during the Sprint Planning Meeting and as needed throughout the Sprint. (SCHWABER & SUTHERLAND, 2011, p.9)

O dono do produto pode estar presente durante a segunda parte do *Sprint Planning Meeting* para ajudar no entendimento dos itens selecionados do *Product Backlog* e negociar os itens do *Sprint Backlog* caso a equipe determine que tenha pouco trabalho ou trabalho excessivo. Outras pessoas podem ser convidadas para a reunião de planejamento para prover conselhos de ordem técnica ou específica (SCHWABER & SUTHERLAND, 2011).

No fim do *Sprint Planning Meeting*, segundo Schwaber e Sutherland (2011) a equipe de desenvolvimento deve ser capaz de explicar ao dono do produto ou ao *Scrum Master* como eles pretendem trabalhar e se organizar para atingir o objetivo do *Sprint* e criar o incremento do produto.

4.3.3 Daily Scrum

O *Daily Scrum*, ou Scrum Diário, como proposto por Schwaber e Sutherland (2011), é um encontro diário de 15 minutos da equipe de desenvolvimento. Seu objetivo é sincronizar as expectativas e atividades de cada componente da equipe e criar um plano para as próximas 24 horas. Durante o Scrum diário é revisado o que foi feito desde o encontro anterior e previsto o que poderá ser feito até o próximo, e cada membro deve apresentar:

- O que foi feito desde o último encontro;

- O que será feito até o próximo encontro;
- E que obstáculos se encontram no caminho.

"The Development Team uses the Daily Scrum to assess progress toward the Sprint Goal and to assess how progress is trending toward completing the work in the Sprint Backlog" (SCHWABER & SUTHERLAND, 2011, p.10).

O Scrum Diário, segundo Schwaber e Sutherland (2011), aumenta a probabilidade da equipe de desenvolvimento atingir o objetivo do *Sprint*, acompanhando, replanejando e ajustando o resto do trabalho a ser efetuado até o fim do *Sprint*. "Every day, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work together as a self-organizing team to accomplish the goal and create the anticipated increment in the remainder of the Sprint" (SCHWABER & SUTHERLAND, 2011, p.10).

O *Scrum Master* tem um papel importante no Scrum Diário. É ele que garante que o encontro aconteça e que seja feito brevemente (por cerca de 15 minutos), apesar da equipe de desenvolvimento ser responsável pela sua condução e resultado. O *Scrum Master* reforça a regra de que apenas os desenvolvedores participem da reunião (SCHWABER & SUTHERLAND, 2011). "The Daily Scrum is not a status meeting, and is for the people transforming the Product Backlog items into an Increment." (SCHWABER & SUTHERLAND, 2011, p.11)

4.3.4 Sprint Review

O *Sprint Review*, como mostrado por Schwaber e Sutherland (2011), é uma reunião informal e colaborativa entre todos os envolvidos no projeto (incluindo a equipe do Scrum e os *stakeholders*) realizada no final de cada *Sprint*. O Objetivo desse encontro é inspecionar o incremento no produto feito no último *Sprint*, adaptar o *Product Backlog*, caso necessário, e visualizar quais as próximas características que podem ser desenvolvidas.

> During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done. This is an informal meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration. (SCHWABER & SUTHERLAND, 2011, p.11)

Segundo Schwaber e Sutherland (2011) o *Sprint Review* inclui os seguintes elementos:

- O dono do produto identifica o que foi e não foi feito com o resultado do último *Sprint*;
- A equipe de desenvolvimento discute quais os pontos positivos e negativos do *Sprint*, e como os problemas encontrados foram resolvidos;
- A equipe de desenvolvimento apresenta o trabalho feito e responde questões sobre o incremento do produto;
- O dono do produto discute sobre o estado do *Product Backlog*, atualizando características como datas;
- E o grupo inteiro colabora sobre o que deve ser feito depois, servindo como uma valiosa entrada para o próximo *Sprint Planning Meeting*.

O resultado desse encontro é revisar o *Product Backlog* e definir os itens mais prováveis para o próximo *Sprint* ajustando o desenvolvimento para se adequar a novas oportunidades (SCHWABER & SUTHERLAND, 2011).

4.3.5 Sprint Retrospective

O *Sprint Retrospective* é um evento onde a equipe do Scrum tem a oportunidade de avaliar e planejar melhorias a ser implementadas no próximo *Sprint* (SCHWABER & SUTHERLAND, 2011).

Schwaber e Sutherland (2011) mostram que a proposta desse encontro é:

- Avaliar o último *Sprint* no que diz respeito a pessoas, relacionamentos, processos e ferramentas;
- Identificar e ordenar os itens que mais foram bem e as melhorias potenciais;
- E criar um plano para implementar melhorias sobre a maneira que a equipe do Scrum trabalha.

> By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a dedicated event focused on inspection and adaptation. (SCHWABER & SUTHERLAND, 2011, p.12)

4.4 RESUMO

O framework Scrum segue uma sequência de etapas para o desenvolvimento de software. A Figura 17 mostra o fluxo do desenvolvimento de jogos com esse framework.

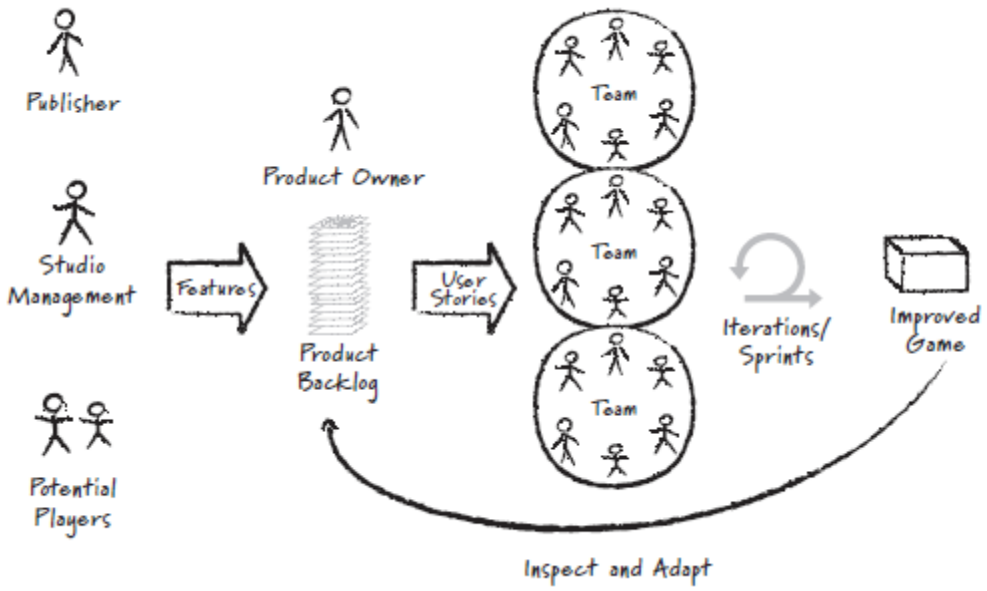

Figura 17 - Agile development flow. (KEITH, 2010, p. 31)

Primeiramente os *stakeholders* avaliam o problema e identificam características e requerimentos para o jogo. Essas características são unidas e listadas no *product backlog*, que é analisada e priorizada pelo *product owner* (dono do produto). Os itens definidos no *product backlog* são conhecidos como *user stories* (histórias do usuário). Elas definem as características do item e o seu valor para o usuário. A equipe (ou equipes) de desenvolvimento trabalha sobre uma ou mais histórias em cada *sprint*, onde, no seu término, serão apresentadas como uma versão melhorada do jogo (KEITH, 2010).

Segundo Keith (2010) existe alguns desafios para que o Scrum funcione. Um (e o maior) deles é o desafio de se aplicar a metodologia devido ao choque com a cultura das empresas. O Scrum cria transparência no desenvolvimento. Se existe alguma deficiência ou ponto negativo, ele é apontado e examinado. “Acting on transparency is the key to the success of applying an agile methodology. Scrum will merely show where and what the problems are. It is up to the individuals, teams, and leaders to solve those problems and thereby realize the benefits of Scrum” (KEITH, 2010, p. 32).

5 ESTUDO DE CASO: SPACE COWBOY

O estudo de caso que será apresentado nesse trabalho trata-se do desenvolvimento de um jogo eletrônico (projeto Space Cowboy) por uma equipe de desenvolvedores de jogos independentes utilizando o Scrum como metodologia de desenvolvimento.

O objetivo da utilização do Scrum nesse projeto é saber se essa metodologia alcança os resultados esperados no desenvolvimento ágil, como mostrado na pesquisa, comparando com os problemas apresentados em metodologias em cascata e espiral.

A equipe de desenvolvimento é nomeada como Forja. Ela é uma equipe que desenvolve jogos independentemente de publicadoras ou empresas. Seu cliente, atualmente, é o cliente final: o próprio jogador. Os componentes do grupo não tem nenhum vínculo com negócios ou empresa, todos desenvolvem jogos como *hobby*, mas todos tem o objetivo de tornar a Forja uma empresa desse ramo.

Como a Forja é independente, muitas das características do Scrum serão abordadas de maneira não convencional. Essas alterações, seus motivos e resultados serão apresentados.

5.1 OBJETIVOS

A Forja pretende, com o jogo Space Cowboy, aumentar o seu portfólio com um jogo simples e de alta qualidade. Não no nível de jogos produzidos por grandes desenvolvedoras, mas em um nível aceitável para um grupo amador.

Esse projeto tem um segundo objetivo: fazer um jogo simples como uma única experiência de jogo com alta qualidade para apresentar a possíveis investidores, com uma proposta de desenvolvimento do mesmo jogo em uma versão mais completa e extensa.

5.2 EQUIPE

A equipe do Scrum para o projeto é bem diversificada. Existem, também, muitas pessoas com mais de uma habilidade e mais de um foco de trabalho.

A seguir segue um quadro com os integrantes e habilidades da equipe.

EQUIPE SCRUM	
PRODUCT OWNER	
Olavo Felipe Souza Silva	
SCRUM MASTER	
Alan Victoria Bazan	
EQUIPE	
Nome	Habilidades
Alan Victoria Bazan	Programação, Arte
Jayson Wesley Cezar Silva	Programação, Game Design
Olavo Felipe Souza Silva	Game Design, Programação
Renan Santos Gomes	Programação, Game Design

Tabela 1 - Equipe do projeto *Space Cowboy* (FORJA ENTERTAINMENT).

5.2.1 Game Designer x Product Owner

Como a equipe de desenvolvimento é independente, não existe nenhum tipo de cliente direto (publicadora, empresa cliente ou empresa de mídia). O cliente do projeto é o cliente final: o jogador. Como não é possível pegar um grupo de jogadores para participar em tempo integral no projeto, a equipe decidiu colocar o *game designer* responsável pelo jogo na função de *product owner*. Ele também faz parte da equipe, mas nada melhor do que colocar, nessa função, a pessoa responsável pela definição do projeto que está mais preocupada em passar a experiência do jogo ao jogador.

5.3 PRODUCT BACKLOG

Antes da proposta do jogo Space Cowboy surgir, o grupo Forja elaborou um evento para discussão de decisão das ideias para jogo, um *brainstorm*. A partir desse evento foram criadas dezenas de propostas de jogo, dentre elas o Space Cowboy.

Esse jogo foi escolhido pela sua simplicidade, claridade na proposta e pela temática aparentemente divertida. Pelas suas características, com base no prévio conhecimento da equipe, esse jogo foi considerado possível em ser construído em dois meses.

Após a escolha do projeto, o dono do produto (também responsável pela ideia) se reuniu à equipe de desenvolvimento para listar as características conhecidas para o jogo. A

partir dessa lista foi criado o *Product Backlog* (Tabela 2), onde contém as próprias características e a sua prioridade para o projeto.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	**Alta**
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	**Alta**
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	**Alta**
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	**Alta**
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	**Média**
O jogador só vence a fase se ele chegar ao seu fim com um número mínimo de vacas, que podem escapar pelo lado do cowboy, ou ser comido por um lobo; e se nenhum lobo alcançar a linha da prancha, caso contrario o jogador perde o jogo imediatamente.	**Média**
O jogo gera fases automaticamente e infinitamente, aumentando a dificuldade (mais vacas e mais lobos).	**Baixa**
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	**Baixa**

Tabela 2 - *Product backlog* inicial do projeto *Space Cowboy* (FORJA ENTERTAINMENT).

É a partir do *Product Backlog* que a equipe do Scrum terá referência para desenvolver o projeto.

5.4 DESENVOLVIMENTO

A partir daqui será apresentado passo-a-passo (por iteração, ou *Sprint*) como se desenrolou o desenvolvimento do projeto Space Cowboy, suas alterações, análises de desempenho, entre outras características.

5.4.1 Iteração 1

5.4.1.1 Sprint Planning

O primeiro *Sprint* do projeto Space Cowboy começou com a reunião chamada *Sprint Planning*. Nessa reunião foram definidas quais seriam os itens do *Product Backlog* a serem desenvolvidos, qual seria o objetivo do *Sprint* e como a equipe de desenvolvimento irá alcançar esse objetivo.

Como o Scrum está sendo usado pela primeira vez pela equipe, foi decidido que o primeiro *Sprint* deveria ser curto para sentir qual o andamento de uma metodologia ágil. O dono do produto deu prioridade ao item de mais importância do *Product Backlog* (chamado de *Story*) e a equipe de desenvolvimento quebrou esse item em tarefas independentes levando em conta a codificação, o *design*, e a arte do jogo para definir a temática, como mostrados na Tabela 3.

SPRINT BACKLOG	
STORIES	
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	
OBJETIVO	
Construir a mecânica base para o jogo: comportamento das vacas.	
FUNCIONALIDADE	**RESPONSÁVEL**
Construir a área ativa de jogo (Cercado).	**Olavo**
Construir a arte e animação da Cerca. Cerca com eletricidade.	**Alan**
Construir o objeto da vaca para ela se mover em qualquer ângulo e velocidade. Permitir que o ângulo e a velocidade seja alterada via código e a mesma alterar seu comportamento.	**Jayson**
Construir a arte e animação da Vaca.	**Alan**
Construir interação entre a Vaca e a Cerca para ela quicar.	**Olavo**
Pesquisar ou Gravar som de mugido.	**Renan**
Construir a execução do som de mugido esporadicamente.	**Renan**

Tabela 3 - *Sprint Backlog* do primeiro *Sprint* (FORJA ENTERTAINMENT).

Como mostrado na Tabela 4, a equipe de desenvolvimento contava com apenas uma semana de trabalho. Como a equipe cria jogos independentes de publicadoras e financiadores,

a jornada de trabalho foi reduzida, limitando-se a trabalhar no projeto apenas de Sábado, Domingo e Quarta-Feira, três dias por semana.

INFORMAÇÕES DO SPRINT	
Data de Início	**23/10/2011**
Limite do Sprint	**30/10/2011**

Tabela 4 - Informações do primeiro *Sprint* (FORJA ENTERTAINMENT).

5.4.1.2 Sprint

A equipe de desenvolvimento desenvolveu as atividades do primeiro *Sprint* com sucesso. Não foi desenvolvido apenas protótipos de programação, como esperado pela equipe para o início da produção de um jogo. Características de arte e design andaram paralelamente com as outras áreas de desenvolvimento.

O resultado da semana é apresentado na Figura 18. Todos os aspectos do jogo foram desenvolvidos para o objetivo do *Sprint*: a mecânica da movimentação e interação do personagem da vaca com a cerca foi programada, a arte dos componentes da cena e os efeitos sonoros.

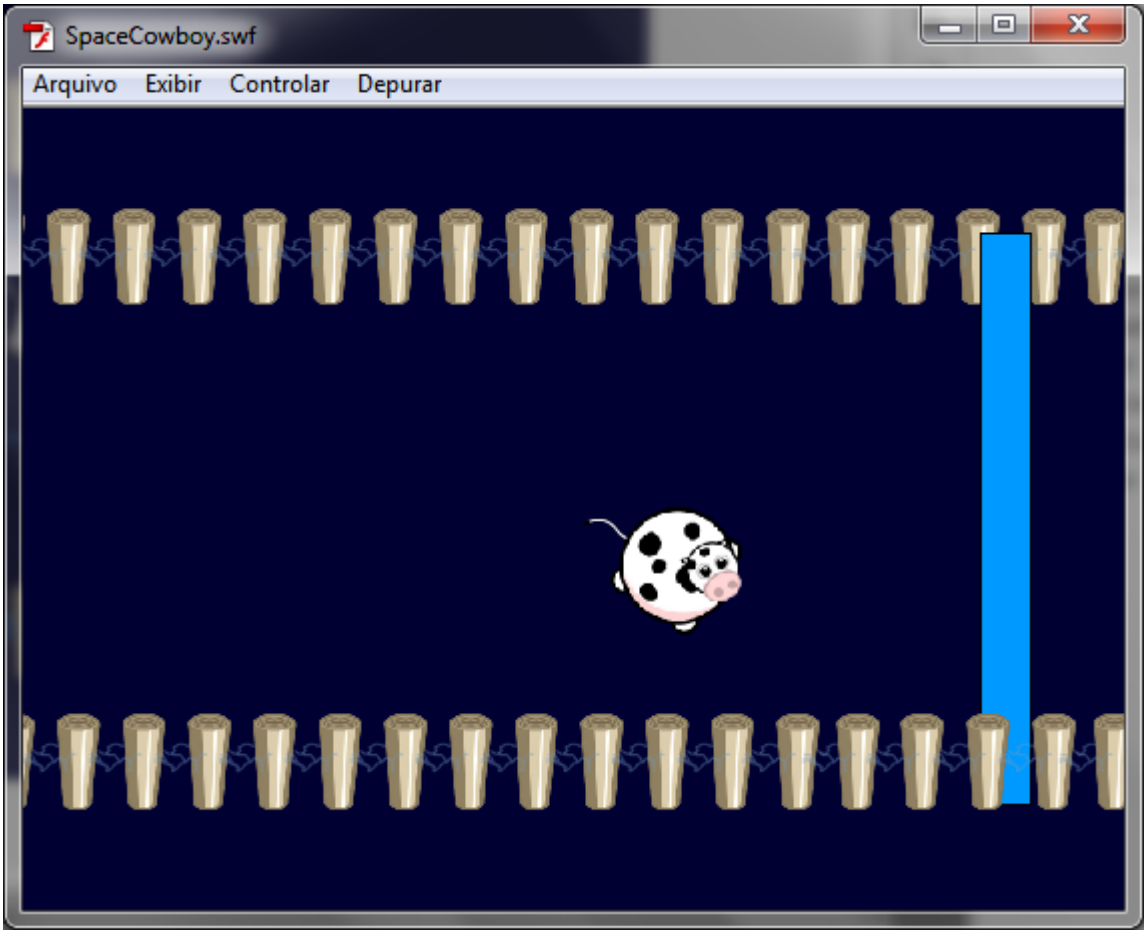

Figura 18 - Produto após o primeiro *Sprint* (FORJA ENTERTAINMENT).

5.4.1.3 Sprint Review

Durante a reunião do *Sprint Review*, a equipe analisou e discutiu os resultados do *Sprint*. O dono do produto aprovou o incremento gerado e considerou o *Sprint* como encerrado. Os resultados podem ser observados na Tabela 5.

SPRINT BACKLOG	
STORIES	
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	
OBJETIVO	
Construir a mecânica base para o jogo: comportamento das vacas.	
FUNCIONALIDADE	**SITUAÇÃO**
Construir a área ativa de jogo (Cercado).	**Finalizado**
Construir a arte e animação da Cerca. Cerca com eletricidade.	**Finalizado**
Construir o objeto da vaca para ela se mover em qualquer ângulo e velocidade. Permitir que o ângulo e a velocidade seja alterada via código e a mesma alterar seu comportamento.	**Finalizado**
Construir a arte e animação da Vaca.	**Finalizado**
Construir interação entre a Vaca e a Cerca para ela quicar.	**Finalizado**
Pesquisar ou Gravar som de mugido.	**Finalizado**
Construir a execução do som de mugido esporadicamente.	**Finalizado**

Tabela 5 - Resultado do primeiro *Sprint* (FORJA ENTERTAINMENT).

Com o término do primeiro *Sprint* novas necessidades foram percebidas pelo dono do produto. Essas necessidades são mais específicas em comparação com os itens que já existiam no *Product Backlog*. Esse documento foi alterado, e sua versão para o segundo *Sprint* é apresentado na Tabela 6, onde os itens sublinhados são as novas necessidades e as linhas da tabela que estiver com o fundo azul são os itens já desenvolvidos.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	**Alta**
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	**Alta**
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	**Alta**
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	**Alta**
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	**Média**
O jogador só vence a fase se ele chegar ao seu fim com um numéro mínimo de vacas, que podem escapar pelo lado do cowboy, ou ser comido por um lobo, e se nenhum lobo alcançar a linha da prancha; caso contrario o jogador perde o jogo imediatamente.	**Média**
Construir arte e animação (diferentes) para um cercado vertical.	**Média**
Animação da vaca girando.	**Média**
Animação da vaca rebatendo (Squash Stretch).	**Média**
Motion Blur na movimentação da cerca.	**Média**
O jogo gera fases automaticamente e infinitamente, aumentando a dificuldade (mais vacas e mais lobos).	**Baixa**
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	**Baixa**
O level design das fases avança com o cenário (background), que tem scroll automático.	**Baixa**

Tabela 6 - Primeira revisão do *Product Backlog* (FORJA ENTERTAINMENT).

5.4.1.4 Sprint Retrospective

Após a primeira experiência da equipe com uma metodologia ágil, foi decidido fazer a reunião de *Sprint retrospective* para relatar as primeiras experiências com o Scrum. Os relatos da equipe são descritos na Tabela 7.

RETROSPECTIVA DO SPRINT	
PONTOS POSITIVOS	
Produtividade elevada.	
Trabalho em conjunto efetivo, causando agilidade no desenvolvimento.	
Trabalho de qualidade.	
Equipe alinhada ao objetivo e foco na situação (contexto) do projeto.	
Boa organização da equipe.	
Resultados visíveis.	
Feedback imediato à problemas, mostrando criatividade dos indivíduos.	
PROBLEMAS	**RESOLUÇÕES**
Falta de preparação para algumas atividades.	**Trabalho em grupo e auxílio ao companheiro; Pesquisa e estudos rápidos e focados.**
Falta de Foco em histórico/lições (técnicas) aprendidas.	**Máxima organização em cima do próprio produto: comentários de códigos, organização das artes, etc., para não quebrar o ritmo do trabalho.**
Atividades dependentes entre pessoas diferentes.	**Trabalho em grupo e auxílio ao companheiro.**

Tabela 7 - Retrospectiva do primeiro *Sprint* (FORJA ENTERTAINMENT).

Apesar de alguns problemas pequenos de organização, a equipe percebeu uma grande produtividade e agilidade no desenvolvimento. O Scrum promoveu um alto nível de comunicação entre os componentes da equipe e agilidade no desenvolvimento, devido ao foco no objetivo do *Sprint*, não no objetivo do projeto. A equipe também percebe o resultado do trabalho rapidamente, mantendo a equipe motivada.

5.4.2 Iteração 2

5.4.2.1 Sprint Planning

No segundo *Sprint Planning*, o dono do produto, em conjunto com a equipe, decidiu o próximo passo do projeto, analisando quais tarefas serão desenvolvidas no segundo *Sprint* e qual o objetivo do mesmo (mostrado na Tabela 8).

SPRINT BACKLOG	
STORIES	
O jogo posiciona bolas (vacas) na tela e elas rebatem pelo campo em ângulos e velocidades diferentes.	
Animação da vaca girando.	
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	
OBJETIVO	
Desenvolver a mecânica de condução das vacas, similar ao jogo Pong.	
FUNCIONALIDADE	**RESPONSÁVEL**
Construir arte e animação (diferentes) para um cercado vertical.	**Alan**
Animação da vaca girando.	**Jayson**
Organizar código atual para se adequar às atividades do sprint.	**Jayson**
Controle do Jogador pelo Teclado da prancha (eixo vertical) e limitação da movimentação.	**Renan**
Construir o desvio da vaca dependendo da posição que a vaca é rebatida.	**Olavo, Jayson**
Construir a arte e animação do Cowboy Espacial.	**Alan**
Construir a execução de efeito sonoro durante a colisão entre vaca e cowboy.	**Olavo, Jayson**

Tabela 8 - *Sprint Backlog* do segundo *Sprint* (FORJA ENTERTAINMENT).

A equipe, junto ao dono do produto, decidiu limitar esse *Sprint* ainda em uma semana, como mostrado na Tabela 9, para manter a equipe focada em um problema ou necessidade menor, e para ela se acostumar com as características do Scrum.

INFORMAÇÕES DO SPRINT	
Data de Início	**23/10/2011**
Limite do Sprint	**30/10/2011**

Tabela 9 - Informações do Segundo *Sprint* (FORJA ENTERTAINMENT).

5.4.2.2 Sprint

O segundo *Sprint* foi desenvolvido, como mostrado na Figura 19, finalizando uma das mecânicas básicas do jogo Space Cowboy. A vaca rebate pelo cercado e o jogador deve rebatê-la de volta não a deixando escapar. Essa mecânica está finalizada, porém, apenas a programação e o design do jogo avançaram. Nenhuma arte prevista para o *Sprint* foi completamente desenvolvida.

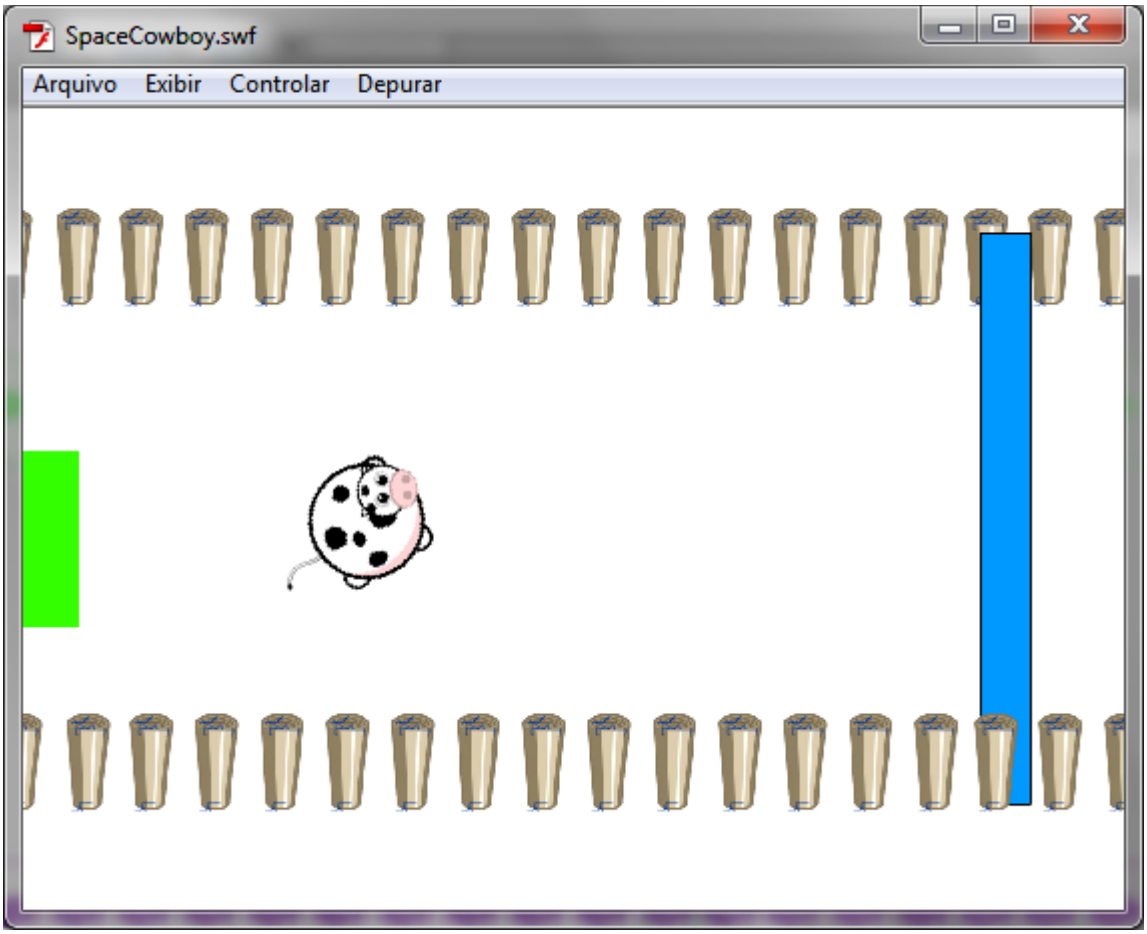

Figura 19 - Produto após o Segundo *Sprint* (FORJA ENTERTAINMENT).

5.4.2.3 Sprint Review

Como apresentado na Tabela 10, todos os itens foram desenvolvidos com exceção dos itens de arte e animação. Todos os membros da equipe têm habilidades diversas dentre a programação e o design de jogos, porém, apenas um possui habilidades de artes visuais. Além de ter atividades relacionadas à arte, esse membro da equipe teve participação na resolução de problemáticas encontradas na programação.

SPRINT BACKLOG	
STORIES	
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	
Animação da vaca girando.	
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	
OBJETIVO	
Desenvolver a mecânica de condução das vacas, similar ao jogo Pong.	
FUNCIONALIDADE	**SITUAÇÃO**
Construir arte e animação (diferentes) para um cercado vertical.	**Não Iniciado**
Animação da vaca girando.	**Finalizado**
Organizar código atual para se adequar às atividades do sprint.	**Finalizado**
Controle do Jogador pelo Teclado da prancha (eixo vertical) e limitação da movimentação.	**Finalizado**
Construir o desvio da vaca dependendo da posição que a vaca é rebatida.	**Finalizado**
Construir a arte e animação do Cowboy Espacial.	**Em Andamento**
Construir a execução de efeito sonoro durante a colisão entre vaca e cowboy.	**Não Iniciado**

Tabela 10 - Resultado do Segundo *Sprint* (FORJA ENTERTAINMENT).

Durante o *Sprint Review* a problemática das artes do jogo foi discutida. Uma solução apresentada foi a possibilidade de inclusão de outra pessoa na equipe com habilidades artísticas.

Depois de ter discutido o resultado do *Sprint*, o dono do produto considerou-o como encerrado, e decidiu colocar os itens de arte nos próximos *Sprints*, considerando que, mesmo sem as artes, o objetivo foi alcançado e a mecânica foi desenvolvida.

O *Product Backlog* foi alterado, sendo inclusos alguns itens para ajustes da mecânica. Segue na Tabela 11 o *Product Backlog* atualizado, com os itens desenvolvidos em cinza e os itens incluídos sublinhados.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	**Alta**
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	**Alta**
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	**Alta**
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	**Alta**
Controle mais preciso de Colisão.	**Alta**
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	**Média**
O jogador só vence a fase se ele chegar ao seu fim com um numéro mínimo de vacas, que podem escapar pelo lado do cowboy, ou ser comido por um lobo, e se nenhum lobo alcançar a linha da prancha; caso contrario o jogador perde o jogo imediatamente.	**Média**
Aceleração e Desaceleração da Prancha (suavização na movimentação)	**Média**
Construir arte e animação (diferentes) para um cercado vertical.	**Média**
Animação da vaca girando.	**Média**
Animação da vaca rebatendo (Squash Stretch).	**Média**
Motion Blur na movimentação da cerca.	**Média**
O jogo gera fases automaticamente e infinitamente, aumentando a dificuldade (mais vacas e mais lobos).	**Baixa**
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	**Baixa**
O level design das fases avança com o cenário (background), que tem scroll automático.	**Baixa**

Tabela 11 - Segunda revisão do *Product Backlog* (FORJA ENTERTAINMENT).

5.4.3 Iteração 3

5.4.3.1 Sprint Planning

Para o terceiro *Sprint*, a equipe do Scrum e o dono do produto resolveram fechar a mecânica base para o jogo. Além da característica desenvolvida no *Sprint* anterior (rebater as vacas mantendo-as no cercado), deve-se desenvolver, em conjunto, o aspecto ação/*shooter*, e testar se essa mecânica é divertida e se é funcional para o objetivo do projeto.

O dono do produto, junto com a equipe de desenvolvimento, decidiu quais itens do *Product Backlog* seriam desenvolvidos e, logo em seguida, a equipe quebrou esses itens em atividades menores para distribuir entre os integrantes. Essas informações são apresentadas na Tabela 12.

SPRINT BACKLOG	
STORIES	
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	
Controle mais preciso de Colisão.	
OBJETIVO	
Desenvolver a segunda mecânica do jogo: aspecto de shooter (jogo de tiro).	
FUNCIONALIDADE	**RESPONSÁVEL**
Construir arte e animação (diferentes) para um cercado vertical.	**Alan**
Construir a arte e animação do Cowboy Espacial.	**Alan**
Construir a execução de efeito sonoro durante a colisão entre vaca e cowboy.	**Olavo**
Implementar uma lógica para aparição de coiotes, que se movem para a esquerda com velocidade configurável e em quantidade e intervalos configuráveis.	**Jayson**
Implementar a relação de colisão entre o coiote e a vaca. Caso a vaca colida com o coiote a mesma é destruída.	**Jayson**
Implementar o tiro do cowboy.	**Renan**
Implementar a lógica de colisão entre o tiro e o coiote. Caso o tiro colida com o coiote o coiote é destruído.	**Renan**
Implementar controle de colisão mais precisa, pelo círculo da vaca, não pela quadrado em volta dela.	**Jayson**
Construir a arte do coiote.	**Alan**
Construir a animação do coiote.	**Alan**
Teste de Design/Mecânica/Diversão	**Grupo**

Tabela 12 - *Sprint Backlog* do terceiro *Sprint* (FORJA ENTERTAINMENT).

Como a equipe de desenvolvimento conseguiu desenvolver a primeira metade da mecânica base em uma semana, a entrega da outra metade ficou combinada para a semana seguinte. As datas são apresentadas na Tabela 13.

INFORMAÇÕES DO SPRINT	
Data	**30/10/2011**
Limite do Sprint	**06/11/2011**

Tabela 13 - Informações do terceiro *Sprint* (FORJA ENTERTAINMENT).

5.4.3.2 Sprint

O terceiro *Sprint* foi desenvolvido sem muitos problemas. Assim como a programação e o *design*, a arte avançou consideravelmente, desenvolvendo as tarefas desse *Sprint*, e as previstas no *Sprint* anterior. O resultado foi a união das duas mecânicas base completa, envolvendo também conteúdo gráfico e animado (Figura 20).

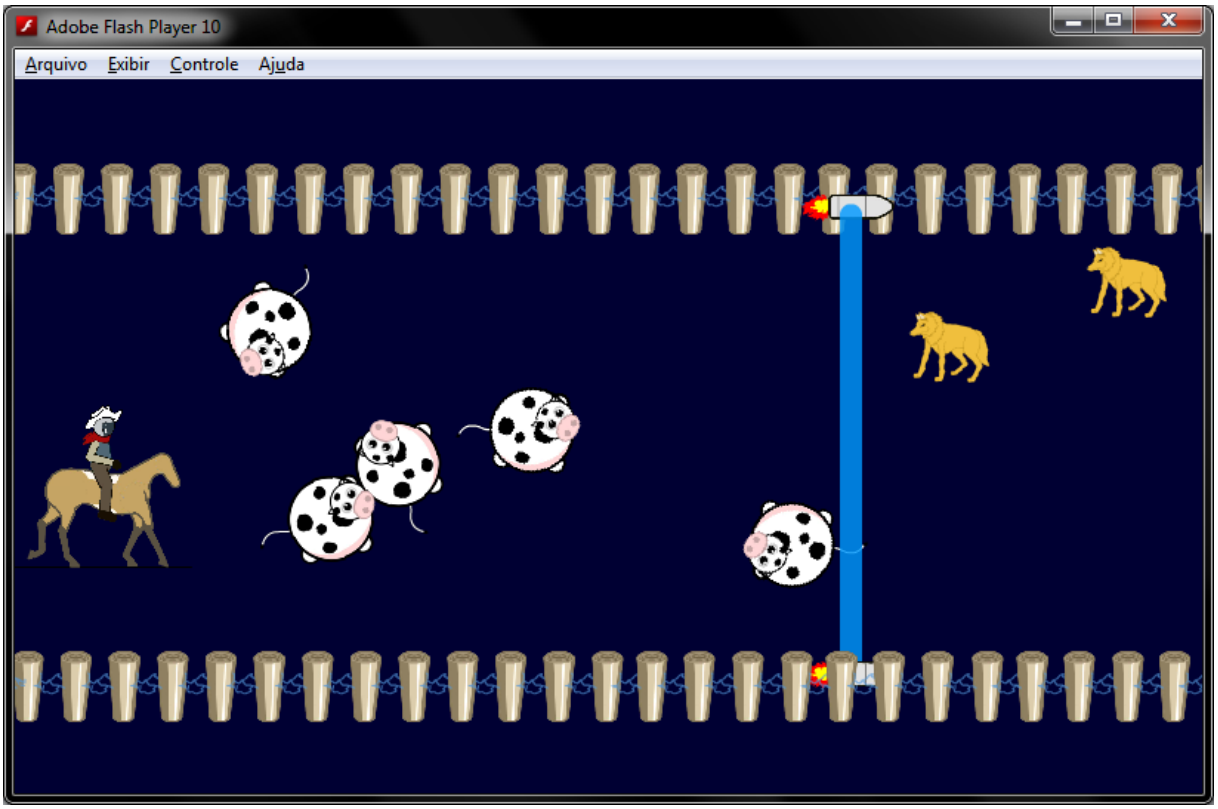

Figura 20 - Produto após o terceiro *Sprint* (FORJA ENTERTAINMENT).

5.4.3.3 Sprint Review

O dono do produto considerou o incremento do *Sprint* totalmente válido em relação aos objetivos do *Sprint* e aos objetivos do projeto. O jogo, até o momento, apresentou corretamente a ideia inicial, além de confirmar que o jogo é promissor no quesito diversão.

Em três semanas de desenvolvimento (com apenas três dias de trabalho por semana) a equipe conseguiu comprovar a qualidade do jogo e achar os pontos divertidos para o jogo, tendo uma visão mais clara de como continuar o desenvolvimento a partir do estado atual.

O resultado do *Sprint* é mostrado na Tabela 14.

SPRINT BACKLOG	
STORIES	
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	
Controle mais preciso de Colisão.	
OBJETIVO	
Desenvolver a segunda mecânica do jogo: aspecto de shooter (jogo de tiro).	
FUNCIONALIDADE	**SITUAÇÃO**
Construir arte e animação (diferentes) para um cercado vertical.	**Finalizado**
Construir a arte do Cowboy Espacial.	**Finalizado, mas marcado para alteração**
Construir a execução de efeito sonoro durante a colisão entre vaca e cowboy.	**Finalizado**
Implementar uma lógica para aparição de coiotes, que se movem para a esquerda com velocidade configurável e em quantidade e intervalos configuráveis.	**Finalizado**
Implementar a relação de colisão entre o coiote e a vaca. Caso a vaca colida com o coiote a mesma é destruída.	**Finalizado**
Implementar o tiro do cowboy.	**Finalizado**
Implementar a lógica de colisão entre o tiro e o coiote. Caso o tiro colida com o coiote o coiote é destruído.	**Finalizado**
Implementar controle de colisão mais precisa.	**Finalizado**
Construir a arte do coiote.	**Finalizado, mas marcado para alteração**
Teste de Design/Mecânica/Diversão	**Finalizado**

Tabela 14 - Resultado do terceiro *Sprint* (FORJA ENTERTAINMENT).

Apesar do sucesso do terceiro *Sprint*, a equipe percebeu que, a partir do quarto a arte será mais requisitada, e não será possível desenvolve-la em grandes quantidades com apenas um indivíduo trabalhando. Para resolver essa problemática, o grupo do projeto convidou outra pessoa, que trabalhava em outro projeto (do próprio grupo Forja), para participar do projeto Space Cowboy, deixando a equipe com a seguinte composição:

EQUIPE SCRUM	
PRODUCT OWNER	
Olavo Felipe Souza Silva	
SCRUM MASTER	
Alan Victoria Bazan	
EQUIPE	
Nome	**Habilidades**
Alan Victoria Bazan	**Programação, Arte**
Jayson Wesley Cezar Silva	**Programação**
Johnny William Cezar Silva	***Arte, Design, Game Design***
Olavo Felipe Souza Silva	**Game Design, Programação**
Renan Santos Gomes	**Programação**

Tabela 15 - Nova formação da equipe (FORJA ENTERTAINMENT).

A equipe do *Scrum* discutiu alguns pontos do incremento do produto e, com o consentimento do dono do produto, alguns itens foram adicionados e alterados como incertos no *Product Backlog*. Os itens adicionados estão sublinhados e os ignorados até o momento marcados como incertos na Tabela 16.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	Alta
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	Alta
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	Alta
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	Alta
Controle mais preciso de Colisão.	Alta
O jogador perde se ele perder todas as vacas, que podem escapar pelo lado do cowboy ou serem comidas por um lobo, ou se algum lobo alcançar a linha do cowboy.	Média
Fase única e infinita, aumentando a dificuldade.	Média
Balanceamento do Jogo em relação aos inimigos e quantidade de tiros.	Média
Implementar Sistema de Pontuação.	Média
Construir arte e animação (diferentes) para um cercado vertical.	Média
Animação da vaca girando.	Média
Incluir Itens que mudam os status do jogo.	Média-Baixa
Aceleração e Desaceleração da Prancha (suavização na movimentação)	Média-Baixa
Animação da vaca rebatendo (Squash Stretch).	Média-Baixa
Motion Blur na movimentação da cerca.	Média-Baixa
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	Baixa
O level design das fases avança com o cenário (background), que tem scroll automático.	Baixa
Animação do Cowboy atirando.	Baixa
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	Incerto(Média)
Animação do Coiote comendo a Vaca	Incerto(Baixa)

Tabela 16 - Terceira revisão do *Product Backlog* (FORJA ENTERTAINMENT).

5.4.4 Iteração 4

5.4.4.1 Sprint Planning

O início do quarto *Sprint* aconteceu com o primeiro *Sprint Planning* após a confirmação da mecânica do jogo. A partir daqui os itens do *Product Backlog* envolvem os itens incluídos pela percepção da equipe durante os *Sprints* anteriores.

A equipe e o dono do produto decidiram o objetivo do próximo *Sprint*. Como a mecânica base já está definida, o objetivo do *Sprint* quatro é a refinação e complemento dela, melhorando o fluxo do jogo, pontuação e condições de derrota, como mostrado na Tabela 17.

SPRINT BACKLOG	
STORIES	
O jogador perde se ele perder todas as vacas, que podem escapar pelo lado do cowboy ou serem comidas por um lobo, ou se algum lobo alcançar a linha do cowboy.	
Fase única e infinita, aumentando a dificuldade.	
Balanceamento do Jogo em relação aos inimigos e quantidade de tiros.	
Implementar Sistema de Pontuação.	
OBJETIVO	
Fluxo do jogo, pontuação e condições de derrota.	
FUNCIONALIDADE	**RESPONSÁVEL**
Manter controle da quantidade de vacas.	**Renan**
Implementar tela básica de game over com botão de recomeço.	**Jayson**
Implementar Game Over ao se perder todas as vacas.	**Renan**
Implementar Game Over quando o Coiote chegar na prancha.	**Jayson**
Implementar lógica de aumento de dificuldade pela velocidade e respawn dos coiotes.	**Jayson**
Balancear a lógica de tiros.	**Grupo**
Balancear respawn dos coiotes.	**Grupo**
Implementar pontuação ao atingir um coiote.	**Olavo**
Implementar pontuação pela distância percorrida pelo cowboy.	**Olavo**
Implementar multiplicador de pontuação de distância pelo número de vacas remanescentes.	**Olavo**
Finalização da arte do cowboy.	**Johnny, Alan**
Finalização da arte do coiote.	**Johnny, Alan**
Construir as fontes para o jogo (estilo velho oeste com placas de madeira no fundo)	**Johnny, Alan**
Implementar lógica para impressão de texto pelo HUD (pontuação, número de vacas e quilometragem).	**Jayson**
Balanceamento das vacas e do cowboy pelo tamanho.	**Grupo**
Implementação dos sons (cowboy atira, coiote entrando, coiote come vaca, coiote come cowboy, galope)	**Olavo**

Tabela 17 - *Sprint Backlog* do quarto *Sprint* (FORJA ENTERTAINMENT).

Como serão inseridas características novas e complexas para o jogo, principalmente no que diz respeito à programação, como controle do estado e fluxo do jogo, sistema de pontuação, HUD (*head up display*), que são interfaces de informações que o jogador recebe enquanto joga, o *Sprint* durará duas semanas (Tabela 18). Para acompanhar o ritmo da programação, foi aproveitado para que, nesse *Sprint*, as artes já existentes fossem redesenhadas ou refinadas.

INFORMAÇÕES DO SPRINT	
Data	**06/11/2011**
Limite do Sprint	**20/11/2011**

Tabela 18 - Informações do quarto *Sprint* (FORJA ENTERTAINMENT).

5.4.4.2 Sprint

Durante o *Sprint* muitas coisas foram desenvolvidas. As artes do cowboy e do coiote foram refeitas, muitas características foram programadas, o HUD foi desenvolvido, incluindo a visualização da distância percorrida, dos pontos, da quantidade de vacas e da quantidade de tiros restantes. O incremento do produto pode ser visualizado na Figura 21.

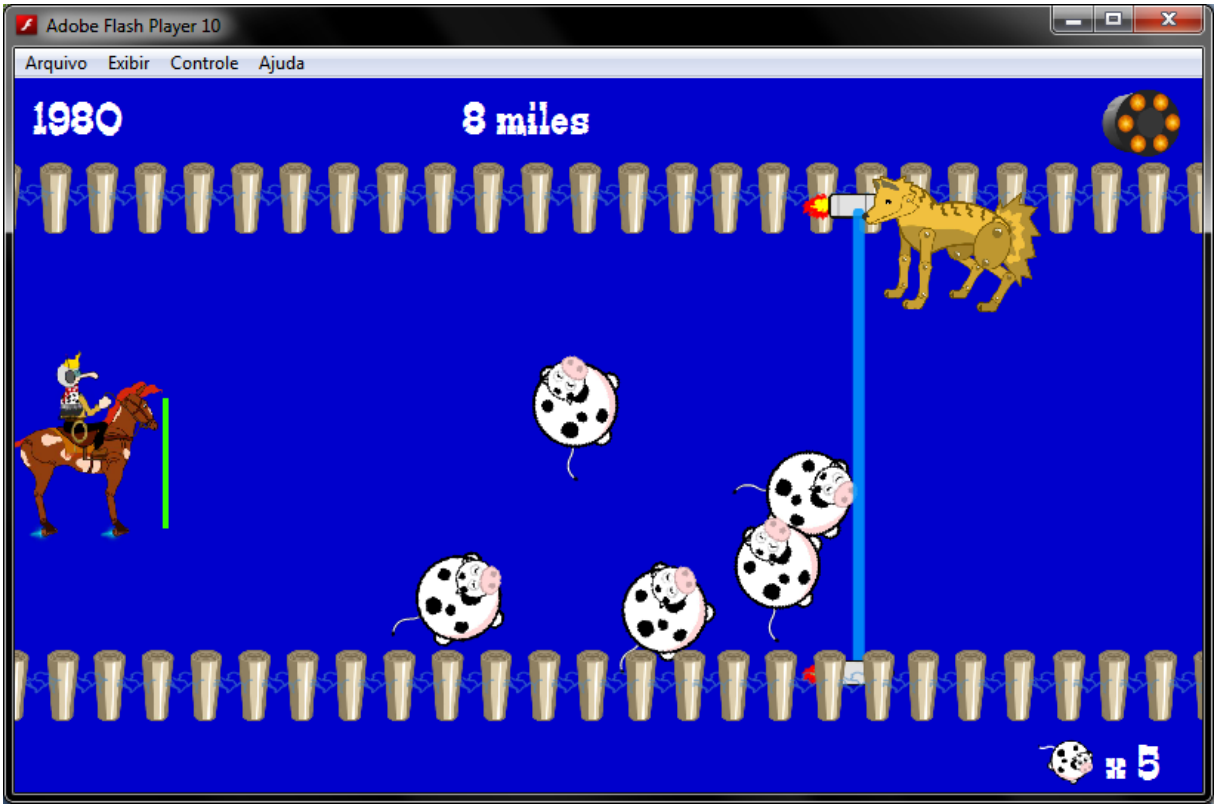

Figura 21 - Produto após o quarto *sprint* (FORJA ENTERTAINMENT).

5.4.4.3 Sprint Review

Como mostrado na Tabela 19, todos os itens planejados para o *Sprint* quatro foram desenvolvidos. O resultado, como visto anteriormente, foi o desenvolvimento completo da jogabilidade. O jogo em si está completo. Todos os itens previstos na ideia do jogo foram desenvolvidos.

SPRINT BACKLOG	
STORIES	
O jogador perde se ele perder todas as vacas, que podem escapar pelo lado do cowboy ou serem comidas por um lobo, ou se algum lobo alcançar a linha do cowboy.	
Fase única e infinita, aumentando a dificuldade.	
Balanceamento do Jogo em relação aos inimigos e quantidade de tiros.	
Implementar Sistema de Pontuação.	
OBJETIVO	
Fluxo do jogo, pontuação e condições de derrota.	
FUNCIONALIDADE	**SITUAÇÃO**
Manter controle da quantidade de vacas.	**Finalizado**
Implementar tela básica de game over com botão de recomeço.	**Finalizado**
Implementar Game Over ao se perder todas as vacas.	**Finalizado**
Implementar Game Over quando o Coiote chegar na prancha.	**Finalizado**
Implementar lógica de aumento de dificuldade pela velocidade e respawn dos coiotes.	**Finalizado**
Alterar a lógica de tiros.	**Finalizado**
Impedir ângulos muito acentuados.	**Finalizado**
Apresentação da quantidade de tiro disponível.	**Finalizado**
Balancear a lógica de tiros.	**Finalizado**
Balancear respawn dos coiotes.	**Finalizado**
Implementar pontuação ao atingir um coiote.	**Finalizado**
Implementar pontuação pela distância percorrida pelo cowboy.	**Finalizado**
Implementar multiplicador de pontuação de distância pelo número de vacas remanescentes.	**Finalizado**
Finalização da arte do cowboy.	**Finalizado**
Finalização da arte do cavalo.	**Finalizado**
Finalização da arte do coiote.	**Finalizado**
Construir as fontes para o jogo (estilo velho oeste)	**Finalizado**
Construir arte do tambor e dos projéteis da pistola como HUD da quantidade de tiros.	**Finalizado**
Implementar lógica para impressão de texto pelo HUD (pontuação, número de vacas e quilometragem).	**Finalizado**
Balanceamento das vacas e do cowboy pelo tamanho.	**Finalizado**

Tabela 19 - Resultado do quarto *Sprint* (FORJA ENTERTAINMENT).

O *Product Backlog* atual (Tabela 20) mostra que o jogo está completo. O incremento resultado do quarto *Sprint* completou todos os itens de alta prioridade para o projeto. O restante dos itens são ajustes da jogabilidade ou o desenvolvimento de características extra jogo, como o menu, animações de apresentação e fim de jogo, cenário do jogo, entre outros.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	Alta
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	Alta
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	Alta
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	Alta
Controle mais preciso de Colisão.	Alta
O jogador perde se ele perder todas as vacas, que podem escapar pelo lado do cowboy ou serem comidas por um lobo, ou se algum lobo alcançar a linha do cowboy.	Média
Fase única e infinita, aumentando a dificuldade.	Média
Balanceamento do Jogo em relação aos inimigos e quantidade de tiros.	Média
Implementar Sistema de Pontuação.	Média
Construir arte e animação (diferentes) para um cercado vertical.	Média
Animação da vaca girando.	Média
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	Média-Baixa
Incluir Itens que mudam os status do jogo.	Média-Baixa
Aceleração e Desaceleração da Prancha (suavização na movimentação)	Média-Baixa
Animação da vaca rebatendo (Squash Stretch).	Média-Baixa
Motion Blur na movimentação da cerca.	Média-Baixa
Construir Ranking dos jogadores.	Média-Baixa
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	Baixa
Tela de Instruções quando o jogo começar.	Baixa
O level design das fases avança com o cenário (background), que tem scroll automático.	Baixa
Animação do Cowboy atirando.	Baixa
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	Incerto(Média)
Animação do Coiote comendo a Vaca	Incerto(Baixa)

Tabela 20 - Quarta revisão do *product backlog* (FORJA ENTERTAINMENT).

5.4.5 Iteração 5

5.4.5.1 Sprint Planning

O planejamento para o *Sprint* cinco foi feito com base no resultado do quarto, que foi a construção completa da mecânica de jogo, incluindo pontuações e apresentação de

informações na tela. Dessa maneira o próximo *Sprint*, como mostrado na Tabela 21, tem como objetivo principal o desenvolvimento completo dos itens que compõem um jogo, extra jogo principal: animação de apresentação, menu de jogo, gravação de ranking, tela de instruções, tela de game over, entre outros.

SPRINT BACKLOG	
STORIES	
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	
Aceleração e Desaceleração da Prancha (suavização na movimentação)	
Animação da vaca rebatendo (Squash Stretch).	
Motion Blur na movimentação da cerca.	
Construir Ranking dos jogadores.	
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	
Tela de Instruções quando o jogo começar.	
O level design das fases avança com o cenário (background), que tem scroll automático.	
Animação do Cowboy atirando.	
OBJETIVO	
Fechar a estrutura e fluxo geral do jogo desde a apresentação até o game over e gravação da pontuação no ranking; e deixar o jogo melhor apresentável.	
FUNCIONALIDADE	**RESPONSÁVEL**
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	Alan
Aceleração e Desaceleração da Prancha (suavização na movimentação)	Renan
Animação da vaca rebatendo (Squash and Stretch).	Alan
Motion Blur na movimentação da cerca.	Renan
Construir salvamento de ranking offline.	Jayson
Construir tela de Ranking.	Grupo
Construir tela de game over.	Grupo
Construir sequência de apresentação	Grupo
Construir tela de créditos.	Grupo
Construir tela inicial com novo jogo, ranking, créditos e habilitação de áudio.	Grupo
Construir tela de input de nome para ranking.	Grupo
Construir tela de instruções.	Grupo
Construção do Background infinito com referências às temáticas espaciais e western.	Johnny
Animar o cowboy atirando.	Alan
Animar o cavalo.	Alan
Animar o Coiote.	Alan
Modificar o tambor e os projeteis para ficar mais visível.	Johnny
Adicionar Efeitos Sonoros	Olavo

Tabela 21 - *Sprint backlog* do quinto *sprint* (FORJA ENTERTAINMENT).

A quantidade de trabalho para esse *Sprint* é muito superior comparado aos *Sprints* anteriores. A equipe se sentiu confiante em dar um passo maior e desenvolver uma quantidade maior de atividades em um único *Sprint*, considerando que as necessidades apresentadas no *Sprint Backlog* são claras para o projeto.

Para desenvolver todas as características a equipe decidiu tornar esse, um *Sprint* de duas semanas, como mostrado na Tabela 22.

INFORMAÇÕES DO SPRINT	
Data	**20/11/2011**
Limite do Sprint	**04/12/2011**

Tabela 22 - Informações do quinto *sprint* (FORJA ENTERTAINMENT).

5.4.5.2 Sprint

Muitos itens do jogo foram desenvolvidos durante o quinto *Sprint*. O incremento do produto produzido contém as seguintes características:

- Tela de apresentação com uma animação simples apresentando a equipe, o autor das músicas incluídas no jogo, e o personagem central do jogo, a Cowball (Figura 22);

Figura 22 - Apresentação do Jogo (FORJA ENTERTAINMENT).

- O menu principal do jogo com botões para jogar, visualizar o ranking e visualizar os créditos do jogo (Figura 23);

Figura 23 - Menu inicial do jogo (FORJA ENTERTAINMENT).

- Tela de instruções, ensinando o jogador a jogar antes do jogo começar (Figura 24);

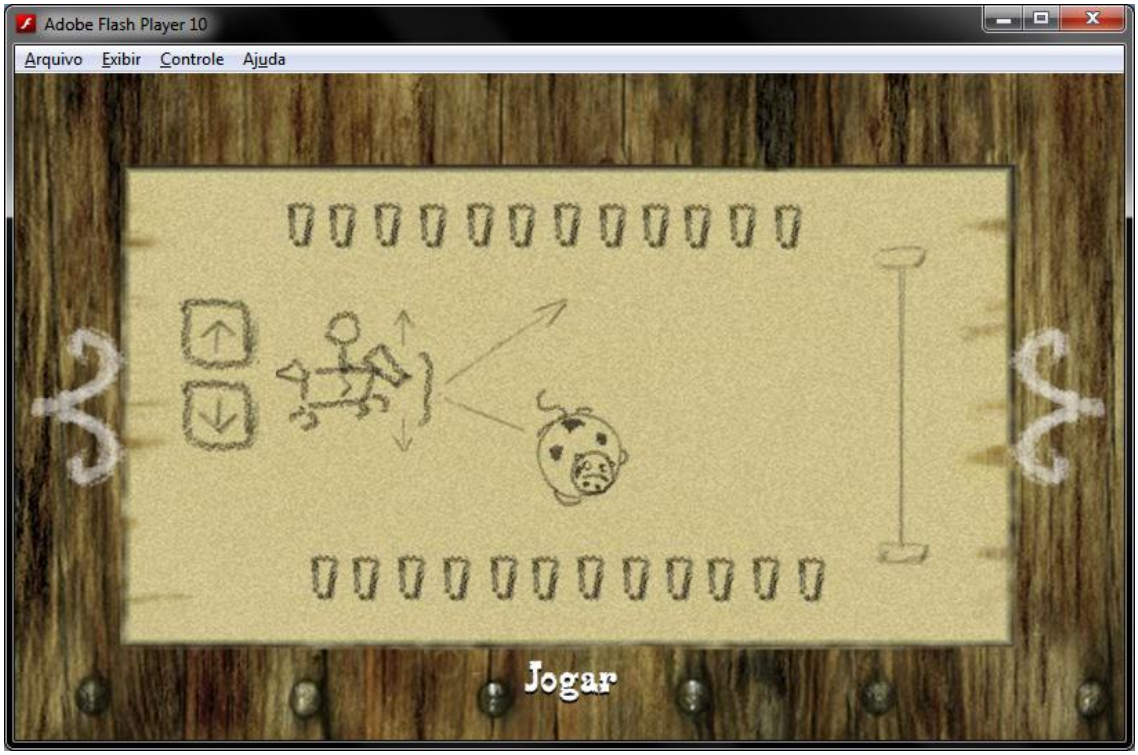

Figura 24 - Tela de instruções (FORJA ENTERTAINMENT).

- Melhoria e detalhamento do ambiente do jogo, adicionando música, animações do cowboy quando atira, plano de fundo com planetas e referências do tema (Figura 25);

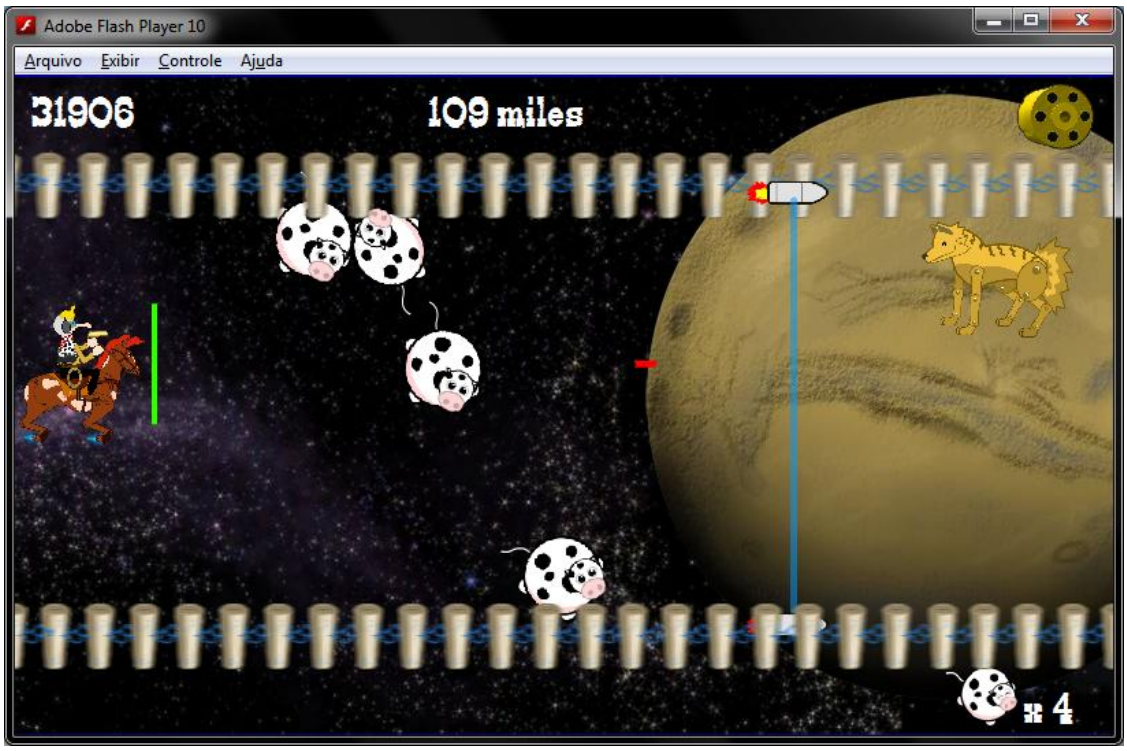

Figura 25 - Tela de jogo (FORJA ENTERTAINMENT).

- Quadro para gravação da pontuação do jogador no ranking (Figura 26);

Figura 26 - Gravação da pontuação (FORJA ENTERTAINMENT).

- E tela de ranking com a gravação da pontuação do jogador (Figura 27).

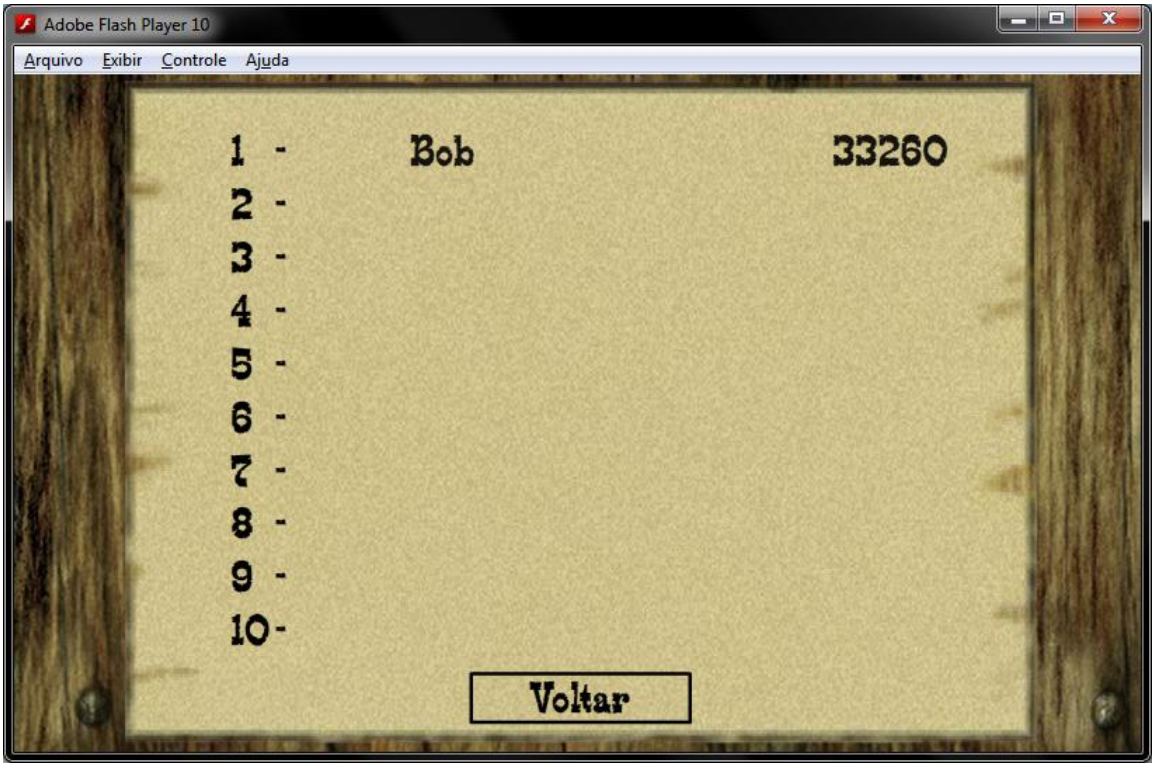

Figura 27 - Tela de Ranking (FORJA ENTERTAINMENT).

5.4.5.3 Sprint Review

No *Sprint Review* o incremento do produto foi apresentado e discutido. A equipe finalizou o *Sprint* e o dono do produto considerou que o seu objetivo foi cumprido. A Tabela 23 mostra o estado das atividades previstas no *Sprint Backlog*. O incremento possui algumas pendências, porém o *Sprint* foi dado como finalizado e essas pendências serão desenvolvidas no sexto, e último, *Sprint*, junto aos outros detalhes que serão melhorados.

SPRINT BACKLOG	
STORIES	
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	
Aceleração e Desaceleração da Prancha (suavização na movimentação)	
Animação da vaca rebatendo (Squash Stretch).	
Motion Blur na movimentação da cerca.	
Construir Ranking dos jogadores.	
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	
Tela de Instruções quando o jogo começar.	
O level design das fases avança com o cenário (background), que tem scroll automático.	
Animação do Cowboy atirando.	
OBJETIVO	
Fechar a estrutura e fluxo geral do jogo desde a apresentação até o game over e gravação da pontuação no ranking; e deixar o jogo melhor apresentável.	
FUNCIONALIDADE	**SITUAÇÃO**
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	Finalizado
Desaceleração da Prancha (suavização na movimentação)	Finalizado
Animação da vaca rebatendo (Squash Stretch).	Finalizado
Motion Blur na movimentação da cerca.	Finalizado
Construir salvamento de ranking offline.	Finalizado
Construir tela de Ranking.	Finalizado
Constrir tela de game over.	Finalizado
Construir sequencia de apresentação	Finalizado
Construir tela de créditos.	Finalizado, mas sem conteúdo
Construir tela inicial com new game, rankin e créditos.	Finalizado
Habilitação de Audio	Pendente
Construir tela de input de nome para ranking.	Finalizado
Construir tela de instruções.	Finalizado
Construção do Background infinito (com easter eggs).	Finalizado
Animar o cowboy atirando.	Finalizado
Animar o cavalo.	Finalizado
Animar o Coiote.	Pendente
Modificar o tambor e os projeteis para ficar mais visível.	Finalizado
Adicionar Efeitos Sonoros	Pendente

Tabela 23 - Resultado do quinto *sprint* (FORJA ENTERTAINMENT).

O dono do produto, junto à equipe de desenvolvimento adicionou algumas características, e fez alterações ao *product backlog* para o último *Sprint* e para a finalização do jogo. Essas novas características estão apresentadas sublinhadas na Tabela 24.

PRODUCT BACKLOG	
STORY	**PRIORIDADE**
O jogo posiciona bolas (vacas) na tela e elas ficam rebatendo pelo campo em ângulos e velocidades diferentes.	**Alta**
O jogador controla uma prancha (cowboy montado) que impede as vacas de irem para a esquerda - único lado que elas não devem rebater. Quanto mais na quina da prancha a bola bater, maior será seu desvio do ângulo de incidência.	**Alta**
De tempos em tempos o jogo lança inimigos - lobos espaciais - na direção da prancha em linha reta. Caso o lobo entre em contato com uma vaca, a vaca é perdida.	**Alta**
Jogador que controla a prancha pode atirar a partir dela nos lobos. O tiro se move na direção dos lobos em linha reta na altura em que a prancha se encontra, como um jogo de tiro. Caso o tiro colida com o lobo, ele é destruído.	**Alta**
Controle mais preciso de Colisão.	**Alta**
O jogador perde se ele perder todas as vacas, que podem escapar pelo lado do cowboy ou serem comidas por um lobo, ou se algum lobo alcançar a linha do cowboy.	**Média**
Fase única e infinita, aumentando a dificuldade.	**Média**
Balanceamento do Jogo em relação aos inimigos e quantidade de tiros.	**Média**
Implementar Sistema de Pontuação.	**Média**
Construir arte e animação (diferentes) para um cercado vertical.	**Média**
Animação da vaca girando.	**Média**
Mudar condição de game over do lobo para acionar apenas se o lobo colide com o cowboy	**Média-Baixa**
Aceleração e Desaceleração da Prancha (suavização na movimentação)	**Média-Baixa**
Animação da vaca rebatendo (Squash Stretch).	**Média-Baixa**
Motion Blur na movimentação da cerca.	**Média-Baixa**
Construir Ranking dos jogadores.	**Média-Baixa**
O jogo deve ter uma tela de apresentação antes do jogo e uma tela de fim de jogo (antes da tela de apresentação ser mostrada novamente).	**Baixa**
Tela de Instruções quando o jogo começar.	**Baixa**
O level design das fases avança com o cenário (background), que tem scroll automático.	**Baixa**
Animação do Cowboy atirando.	**Baixa**
Colocar todos os efeitos sonoros.	**Baixa**
Construir Conteúdo da tela de créditos	**Baixa**
Construir Conteúdo da tela de instruções	**Baixa**
Construção de um botão para Ligar e Desligar o som.	**Baixa**
Balanceamento do Jogo.	**Baixa**
Animação do Coiote e do Cavalo, e de mortes do coiote e da vaca.	**Baixa**
Informação de Pré-jogo e Pós-Jogo (contagem regressiva e informação de game over).	**Baixa**
Construção do Pré-Loader.	**Baixa**
Implementar Reload periódico.	**Baixa**

Incluir Itens que mudam os status do jogo.	**Incerto(Média-Baixa)**
A fase dura até que o jogador atravesse certa distância (simulada por uma barra de progresso que avança com o tempo).	**Incerto(Média)**
Animação do Coiote comendo a Vaca	**Incerto(Baixa)**

Tabela 24 - Quinta revisão do *product backlog* (FORJA ENTERTAINMENT).

5.4.6 Iteração 6

5.4.6.1 Sprint Planning

O sexto e último *sprint* deu início com o jogo praticamente completo. Todas as características previstas para se construir o jogo completo, desde a tela de apresentação até o fim do jogo, foram desenvolvidas, restando para esse *sprint* apenas ajustes de mecânica e refinação das artes e efeitos sonoros.

Para a construção do último *Sprint Backlog* (Tabela 25), todos os itens restantes do *product backlog* foram incluídos, fora os itens marcados como incertos, que foram definitivamente descartados.

SPRINT BACKLOG	
STORIES	
Colocar todos os efeitos sonoros.	
Construir Conteúdo da tela de créditos	
Construir Conteúdo da tela de instruções	
Construção de um botão para Ligar e Desligar o som.	
Balanceamento do Jogo.	
Animação do Coiote e do Cavalo, e de mortes do coiote e da vaca.	
Informação de Pré-jogo e Pós-Jogo (contagem regressiva e informação de game over).	
Construção do Pré-Loader.	
Implementar Reload periódico.	
OBJETIVO	
Refinação e finalização do Jogo.	
FUNCIONALIDADE	**RESPONSÁVEL**
Construir Conteúdo da tela de créditos.	**Grupo**
Construir Conteúdo da tela de instruções.	**Grupo**
Construir classe para controle de som.	**Alan**
Construir botão para ligar e desligar o som.	**Alan**
Balancear o Jogo.	**Grupo**
Colocar efeitos sonoros.	**Olavo**
Animar o Coiote.	**Alan**
Animar o Cavalo.	**Alan**
Animar a morte da Vaca.	**Johnny**
Animar a morte do Coiote.	**Johnny**
Adicionar informações de Pré-Jogo (contagem regressiva).	**Jayson**
Adicionar Pós-Jogo (game over).	**Jayson**
Desaparecer com o cowboy se ele for atingido por um coiote.	**Alan**
Construção do pré-loader.	**Renan**
Implementar Reload periódico.	**Jayson**
Impedir a vaca de fugir.	**Alan**
Arrumar o fundo e fazer as placas.	**Johnny**

Tabela 25 - *Sprint backlog* do sexto *sprint* (FORJA ENTERTAINMENT).

Para o desenvolvimento do último *Sprint* a equipe de desenvolvimento, junto ao dono do produto, definiram um prazo de 2 semanas (Tabela 26). Mesmo esse *sprint* sendo apenas para refinação, há ainda muito trabalho a ser feito, principalmente em relação à aparência do jogo, cujo refinamento é trabalhoso.

INFORMAÇÕES DO SPRINT	
Data	**04/12/2011**
Limite do Sprint	**18/12/2011**

Tabela 26 - Informações do sexto *sprint* (FORJA ENTERTAINMENT).

5.4.6.2 Sprint

O resultado desse último *sprint* foi excepcional: os efeitos sonoros e músicas foram todas implementadas no jogo, os créditos foram desenvolvidos (Figura 28) e o jogo foi refinado, deixando o modo de jogar mais fluído e divertido.

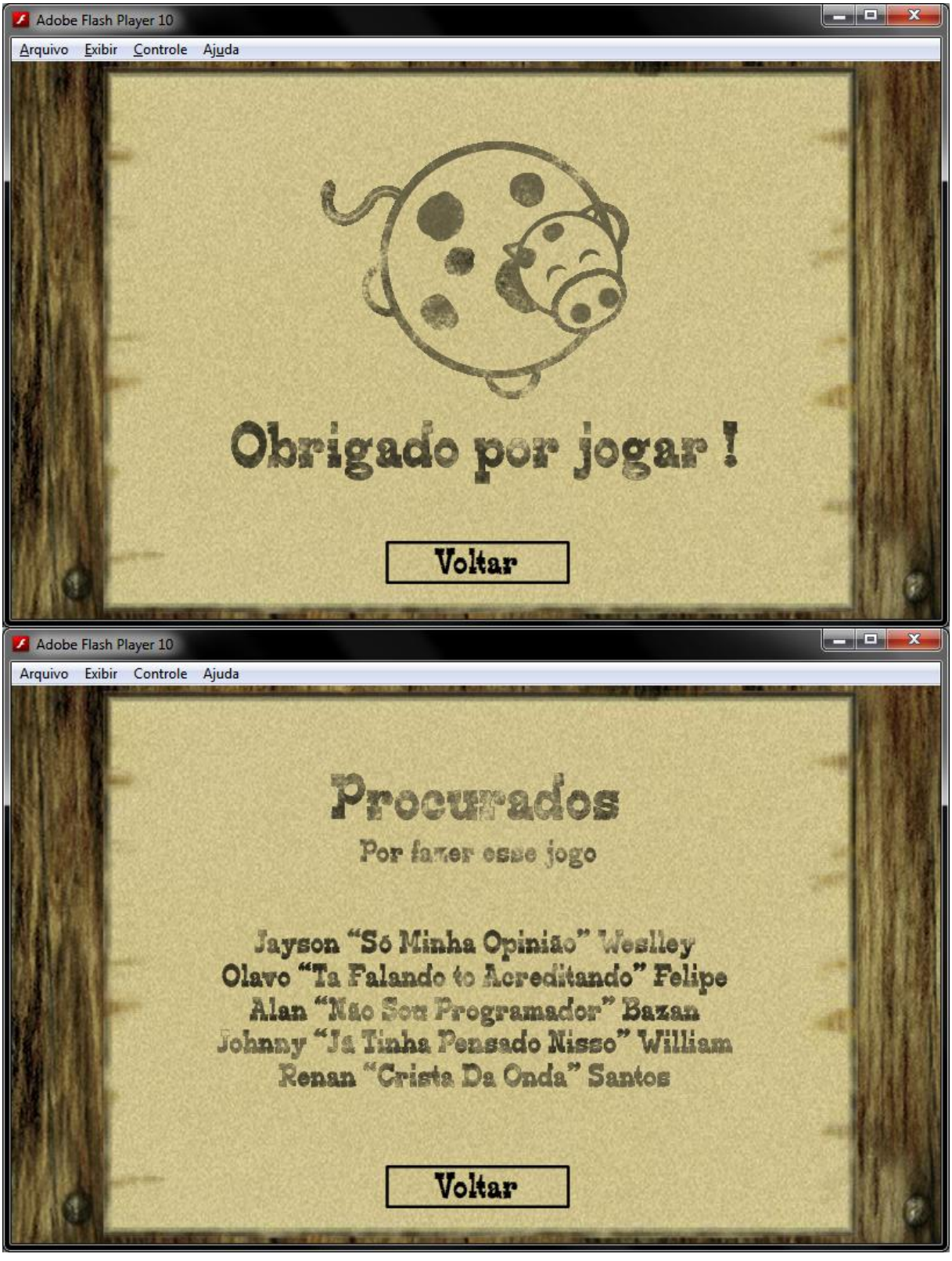

Figura 28 - Tela de créditos (FORJA ENTERTAINMENT).

Foi construída uma tela de instruções mais interessante e refinada, para garantir o entendimento do jogo antes de começá-lo. Foram utilizadas imagens mais relevantes ao jogador e foi adicionado um texto explicativo (Figura 29).

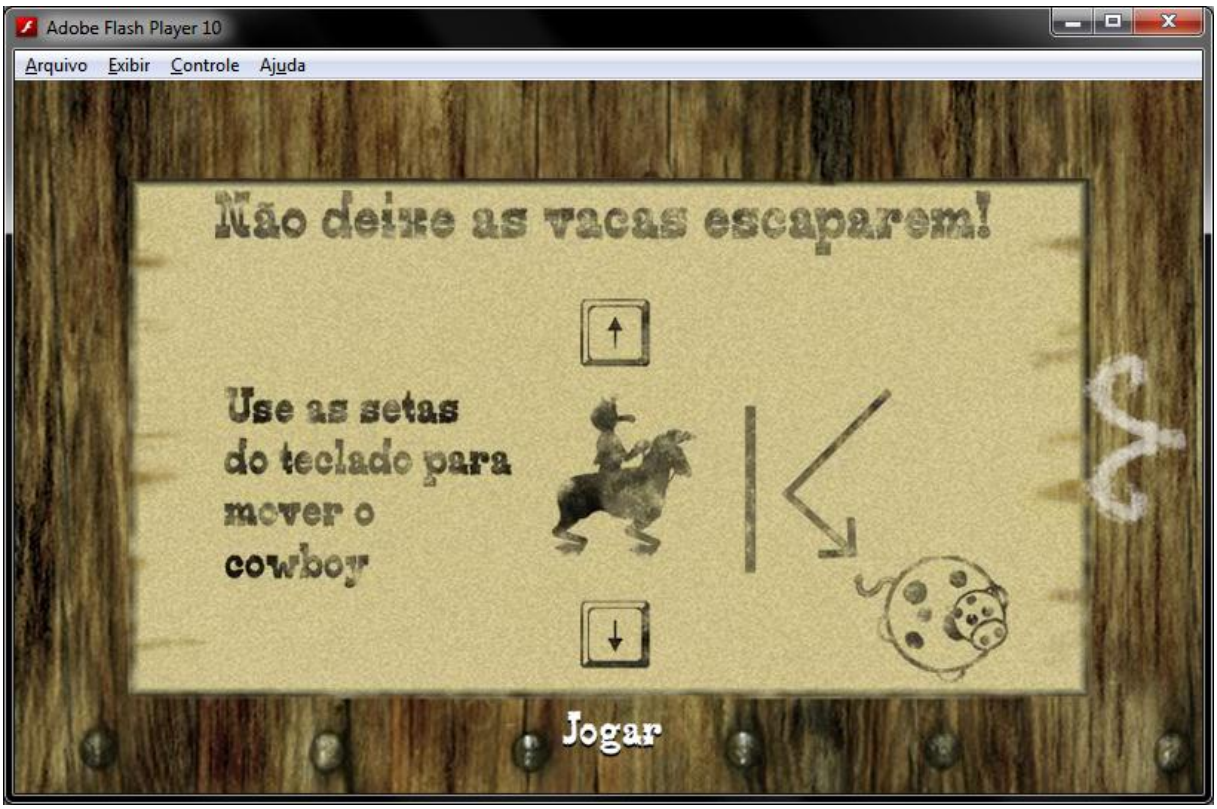

Figura 29 - Nova tela de instruções (FORJA ENTERTAINMENT).

Outra característica desenvolvida foi a adição de controle do som do jogo (Figura 30), permitindo o jogador de deixá-lo mudo ou com a execução dos efeitos sonoros.

Figura 30 - Novo menu inicial (FORJA ENTERTAINMENT).

5.4.6.3 Sprint Review

Durante o *Sprint Review*, o resultado do *sprint* foi apresentado (Tabela 27). Todos os itens previstos para o desenvolvimento foram concluídos sem problemas.

O dono do produto considerou que o projeto atingiu o objetivo esperado e marcou o projeto como finalizado.

SPRINT BACKLOG	
STORIES	
Colocar todos os efeitos sonoros.	
Construir Conteúdo da tela de créditos	
Construir Conteúdo da tela de instruções	
Construção de um botão para Ligar e Desligar o som.	
Balanceamento do Jogo.	
Animação do Coiote e do Cavalo, e de mortes do coiote e da vaca.	
Informação de Pré-jogo e Pós-Jogo (contagem regressiva e informação de game over).	
Construção do Pré-Loader.	
Implementar Reload periódico.	
OBJETIVO	
Refinação e finalização do Jogo.	
FUNCIONALIDADE	**SITUAÇÃO**
Construir Conteúdo da tela de créditos.	Finalizado
Construir Conteúdo da tela de instruções.	Finalizado
Construir classe para controle de som.	Finalizado
Construir botão para ligar e desligar o som.	Finalizado
Balancear o Jogo.	Finalizado
Colocar efeitos sonoros.	Finalizado
Animar o Coiote.	Finalizado
Animar o Cavalo.	Finalizado
Animar a morte da Vaca.	Finalizado
Animar a morte do Coiote.	Finalizado
Adicionar informações de Pré-Jogo (contagem regressiva).	Finalizado
Adicionar Pós-Jogo (game over).	Finalizado
Desaparecer com o cowboy se ele for atingido por um coiote.	Finalizado
Construção do pré-loader.	Finalizado
Implementar Reload periódico.	Finalizado
Impedir a vaca de fugir.	Finalizado
Arrumar o fundo e fazer as placas.	Finalizado

Tabela 27 - Resultado do sexto *sprint* (FORJA ENTERTAINMENT).

5.4.6.4 Sprint Retrospective

Após a apresentação e revisão do *Sprint* a equipe do Scrum se reuniu para efetuar a retrospectiva do *Sprint* para ressaltar algumas percepções que surgiram durante o último *sprint* e a análise do Scrum em geral. O resultado da análise da equipe é mostrado na Tabela 28.

RETROSPECTIVA DO SPRINT	
PONTOS POSITIVOS	
Alta produtividade durante o desenvolvimento, principalmente nos últimos sprints, com a presença de um artista habilidoso.	
Comunição aberta e constante, fazendo com que todos saibam a situação e necessidades do projeto, seja qual for a área.	
Equipe alinhada e com foco nos objetivos dos sprints.	
Boa organização e adaptação natural, mudando a abordagem de acordo com a necessidade	
Resultados visíveis.	
Feedback imediato à problemas, mostrando criatividade dos indivíduos.	
PROBLEMAS	**RESOLUÇÕES**
Falta de preparação para algumas atividades.	**Trabalho em equipe. Percepção imediata da problemática e da pessoa certa para a resolução da mesma.**
Atividades dependentes entre pessoas diferentes.	**Troca de atividades e centralização de tarefas semelhantes em uma única pessoa durante o desenvolvimento conforme as necessidades.**

Tabela 28 - Retrospectiva do último *sprint* (FORJA ENTERTAINMENT).

5.5 RESULTADOS

Após o desenvolvimento do jogo Space Cowboy a equipe Forja o disponibilizou gratuitamente para o público. O jogo foi colocado no site Kongregate dia 17 de Dezembro de 2011.

O Kongregate é um site agregador de jogos gratuitos para se jogar no navegador de internet. Qualquer pessoa que possua um jogo de autoria própria pode submetê-lo para o site. O Kongregate possui uma base de mais de 50 mil jogos de diferentes categorias. O jogo Space Cowboy é um deles e pode ser acessado pelo link http://www.kongregate.com/games/forjaent/space-cowboy.

Desde o dia 17 de Dezembro de 2011 até o dia 12 de Março de 2012 o jogo foi jogado 1187 vezes e possui uma classificação de 2.6 pontos de um total de 5, como mostrado na Figura 31.

Figura 31 - Space Cowboy no Kongregate (KONGREGATE).

6 CONCLUSÃO

O desenvolvimento de software mostra a cada dia ser uma atividade altamente complexa. A tecnologia cresce a uma velocidade incrível, se modernizando cada vez mais rápido.

A produção de um jogo possui essa complexidade presente no desenvolvimento de softwares, porém, essa complexidade é maximizada pelo envolvimento de outras áreas diversas: *design*, arte visual, música, produção de textos e histórias, e áreas que estudam o comportamento humano.

Um aspecto que aumenta drasticamente a complexidade do desenvolvimento do jogo é que o mesmo é produzido para causar uma experiência e satisfazer emocionalmente o jogador através da diversão. A presença do aspecto humano como objetivo central do projeto faz com que o sucesso do projeto seja pouco previsível. É difícil planejar um jogo e os seus efeitos no jogador e ser assertivo.

Nas metodologias em cascata e espiral, como apresentadas nesse trabalho, partem do princípio que o objetivo e o caminho para alcançar esse objetivo já foram planejados. O trabalho desenvolvido nessas metodologias segue o script do planejamento. Mas e se o caminho planejado não for o melhor caminho? E se o jogo criado não produzir o efeito esperado no jogador? Existem estudos de comportamento e conhecimentos que ajudam a prever alguns aspectos humanos, mas eles são mutáveis e variam de acordo com o estereótipo de cada um.

Se uma equipe de desenvolvimento produz jogos para meninas, eles podem abusar da cor rosa em seus jogos, mas será que a utilização da cor rosa é o suficiente para atingir o seu público? Será que, dependendo da temática, um jogo teria um melhor resultado se ele fosse azul, ao invés de rosa? Com metodologias baseadas em seguir à risca o planejamento essas perguntas nunca vão receber uma resposta convincente.

O Scrum, e outras metodologias ágeis, se baseiam na constante transformação do projeto de acordo com as necessidades e com o conhecimento adquirido durante a produção. Nesse caso, essas metodologias seriam ideais para o desenvolvimento de jogos eletrônicos.

A metodologia ágil trabalha em iterações de todo o processo de desenvolvimento, incluindo aquisição e análise de requisitos, planejamento (ou replanejamento), produção, teste

e implantação. A partir desse modelo, o produto final é criado em camadas e, em cada iteração, o planejamento é revisado a partir do desenvolvimento dessa camada, possuindo alguma referência, diferente de um planejamento inicial, que também é importante.

Em relação à industria de jogos eletrônicos, esse tipo de desenvolvimento é essencial para a certeza de sucesso de um jogo. Todo projeto de jogo é mutável, características são incluídas e removidas para que o jogo seja divertido. Porém, nas metodologias em cascata e espiral, essas mudanças não são previstas, causando atrasos e gastos elevados, podendo uma mudança ocorrer em um momento crítico do projeto.

Com a utilização do Scrum essas mudanças são previstas. Toda a equipe tem consciência de que o projeto é pouco conhecido e que ele será moldado para que cada incremento do projeto seja produzido com o maior valor possível. Sendo assim, a cada iteração, o jogo é revisto e analisado, fazendo com que mudanças drásticas no projeto, sejam eles em relação à diversão (aspecto principal), arte, tecnologia ou história, sejam feitas antes do projeto tomar proporções grandiosas, antes que as artes e a mecânica de jogo sejam finalizadas, garantindo que falhas graves de *design* sejam suprimidas no início do projeto. Isso faz com que atrasos inesperados desapareçam, e que o dinheiro gasto no desenvolvimento seja utilizado para a produção de um aspecto que tenha valor para o produto.

Dessa maneira a utilização de metodologias ágeis, não apenas o Scrum, é vantajosa para desenvolvedores de jogos eletrônicos, tanto em aspectos como prazo, custo e qualidade, mas também mostram para investidos e publicadores um resultado visível do projeto logo nas primeiras iterações, diminuindo o cancelamento de projetos a partir das publicadoras e investidores, e diminuindo o desaparecimento dessas empresas e, com ela, sua capacidade, por causa de falhas de projeto.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ADAMS, Ernest; ROLLINGS, Andrew. **Game Design and Development: Fundamentals of Game Design.** Pearson Prentice Hall, United States of America, 2007.

AGILE MANIFESTO. **Princípios por trás do Movimento Ágil**. Disponível em: http://agilemanifesto.org/iso/ptbr/principles.html. Acessado em 10 de Outubro de 2011.

BATES, Bob. **Game Design.** 2ª Edição. Thomson Course Technology PTR, Boston, 2004.

BETHKE, Erik. **Game Development and Production.** Wordware Publishing. Texas. 2003.

BOEHM, Barry W.. **A Spiral Model of SoftwareDevelopment and Enhancement.** TRW Defence System Group, 1988.

BROOKHAVEN HISTORY. **The First Video Game?** Brookhaven National Laboratory. Disponivel em: <http://www.bnl.gov/bnlweb/history/higinbotham.asp>. Acessado em 28 de Dezembro de 2011.

DISCOVERY CHANNEL. **A Era do Videogame**. 2006. Documentário.

FORJA ENTERTAINMENT.

FULLERTON, Tracy. **Game Design Workshop - A Playcentric Approach to Creating Innovative Games.** 2ª Edição. Elsevier. 2008.

IGN. Disponível em: <http://www.ign.com>.

KEITH, Clinton. **Agile Game Development with Scrum.** Addison-Wesley, 2010.

KONGREGATE. Disponível em: < http://www.kongregate.com>.

LAPLANTE, Phillip A.; NEILL, Colin J.. **"The Demise of the Waterfall Model Is Imminent" and Other Urban Myths.** Penn State University, 2004.

RETROSPACE. **A História dos Videogames**. Outerspace Disponível em: <http://outerspace.terra.com.br/retrospace/mainconsoles.htm>. Acessado em 28 de Dezembro de 2011.

ROYCE, Winston W.. **Managing the Development of Large Software Systems.** IEEE WESCON, 1970.

SALEN, Katie; ZIMMERMAN, Eric. **Rules of Play: Game Design Fundamentals.** MIT Press, 2003.

SCHELL, Jesse. **The Art of Game Design: A Book of Lenses.** Morgan Kaufmann Publishers, Burlington, 2008.

SCHWABER, Ken. **SCRUM Development Process.** Proc. of OOPSLA'95 Workshop on Business Object Design and Implementation, Austin, TX, 1995.

SCHWABER, Ken; SUTHERLAND, Jeff. **The Definitive Guide to Scrum: The Rules of the Game**. Scrum.org, 2011.

.