Project Flip Jump

Group 17
Mikael Ahlberg
Daniel Ericsson
Axel Stenkula
Johannes Svensson
Fredrik Vretblad

Contents

1 Preface

1.1 Expected readership

Primarily the stakeholders except the users, such as system architects, developers, resellers and project leaders.

1.2 Version history

Version	Date	Rational	Changes	Authors
1.0	2007/12/20	Initial version.	-	Mikael Ahlberg Daniel Ericsson Axel Stenkula Johannes Svensson Fredrik Vretblad

2 Introduction

2.1 The users and their problems

Our game idea is a 2D side-scrolling game which is based on a very simple idea. The player needs to survive by jumping between blocks and avoid falling down. To make it a little bit more interesting, the screen is moving and forcing the player to move.

Since the game is based on a very simple idea and has no attribute that excludes men or women, it has the potential to attract both. These simple games are usually spread through community sites aimed at a young audience. Because of that, our game is likely to be played by youngsters spending a lot of time on the Internet. We're aiming our game at young people in the age of 15 to 25.

To play this game no earlier game experience is required, but it's likely that our users will have some kind of gaming experience, because these kind of games are usually shared between friends who plays a lot.

We are convinced that our game is played in a short period of time, say a 10 minute period. We're not expecting people to be playing this game for longer periods since other more complex games tend to capture players interest in a way that our game is not expected to do.

Since the game is focused on a short play-time, the game itself needs to be quick-started and throw the player straight into the action. Also, the player is probably already sitting by his or her computer when he or she decides to start playing our game.

Our game is designed to give the user an entertaining time and to let the user rest from other daily problems, such as homework etc. By playing our game the users will feel relaxed and more motivated to return to their previous task. Another reason to play our game is to pass time quickly if the user has some spare time.

2.2 The main uses of the system

The game should be fast to start up and shut down and it will not be possible to save or pause a game. It should also be fun to play and challenging, therefore we will offer different difficulty levels.

2.2.1 Usage narrative one

Dieter, a typical 15 year old German kid, is sitting in front of his computer and doing some serious homework when suddenly his friend Wolfgang sends him a link to a game through Facebook (a community site). Dieter feels like taking a break and opens up the Internet page, downloads the game, and starts it. A new window on his computer pops up where he can choose difficulty level. Dieter chooses easy since it's his first time playing this game. A quick tutorial/text is displayed on the screen telling him how to jump and move around. He presses a specified button to begin the game. He starts to jump between the blocks in the game, and notices how the screen forces him to progress forward at a higher and higher speed. Suddenly the screen flips 90 degrees so he's heading upwards instead of forward. He was quite surprised and almost missed the next block. A bit later he missed the blocks and falls down outside the screen (due to the speed of the screen was too high, he wasn't fast enough). The screen "game over" turns up, followed by a new high score! He enters his name in the high score list. When the game asks him "do you want to play again?" he answers no and the game shuts down. He felt the game was really entertaining and now he feels more motivated to return to his homework.

2.2.2 Usage narrative two

Kajsa, who is a 20 years old Swedish girl, has nothing to do for the moment and wants to do something entertaining. She then remembers a fun game she plays from time to time (our game), and starts it up. She chooses the hard difficulty level, of course, since she wants to have some challenge. She begins to play directly (after pressing a key to start the game) and when the screen flips 90 degrees for the first time she's not surprised. Of course she can play a bit longer than Dieter, but since she chose a harder difficulty the screen moves a lot faster. When she gets "game over" the high score list pops up. This list is locally stored on the computer and she has just beaten her old record. After writing in her name and quitting the program she must let all her friends know about her new high score!

2.3 The environment in which the system is to be used

The system will be used in a home or office environment for private use, because it's a game that we believe people will play during their off-time or when they need to relax from work. Since most people today own a computer, this is the platform that we will focus on for our game. We will use Java to build our game because of it's portability on the computer platform, like PC and MAC. Because of this, users who would like to play our game only has to have a Java-runtime software installed on their system. It will also require a graphics card with OpenGL-support which all cards since the end of 1990 have. Except this, the game itself will not require a high-end system to be playable, since it's a very simple game (construction wise). Because of the simplicity of the game it will be small which also makes it a lot easier to distribute and for friends to share the game between themselves.

2.4 The scope of the system

Since our time is limited during this project, we have to have this in mind when we're planning and building our system. But as you can see in our scope-list bellow, we still have a reasonably amount of functions to be included in our game to make it fun and playable in our limited time.

The scope-list is structured in four different topics: graphics, sound, game rules and misc. We've done this because it becomes much clearer this way and because it's easier to structure our comprehensive work later on.

With accelerated difficulty level, we mean that the rate of which the screen is moving will increase during play time and the frequency of screen flips will also escalate with time.

Graphics scalability is the ability to decrease or increase the quality of the graphics in order to avoid performance drops and have a greater game experience, which is a functionality we won't implement.

2.4.1 In/Out list

Topic	In	Out
Graphics		
- 3D graphics		X
- 2D graphics	X	
- Graphics scalability		X
- Resize-able window	X	
- In-game menu system	X	
Sound		
- In-game sound effects	X	
- In-game music	X	
Game rules		
- Multiplayer		X
- Single player	X	
- Cooperative play		X
- Save game		X
- Pause game		X

- Changeable difficulty level	X	
- Accelerated difficulty level	X	
- Changeable controls		X
Misc		
- Local high score list	X	
- Tutorial text	X	

2.5 Factors to take into account

- When talking about games in general we always have the factor that people don't know how to play the game, a factor that's common for most computer based games. To solve this problem, most game developers sends an manual with the game to help the user get started. We will have a short tutorial in our game and an manual that comes with the game. This manual will not only explain the game and it's settings, it will also include a FAQ-list containing common problems that can occur during play, and how to solve them.

- To minimize problems and user frustration, we're planning to introduce the game to a number of test persons before release. Those persons will then give us important quality feedback so that we can adjust the system and manual according to these results. Another aspect of the tests are to determine if our game is entertaining. To measure the entertainment level, our test persons will rate and comment the game and we will then evaluate the results and make changes if necessary.

- Another important thing that we need to take into account is computer system requirements to run this game, with a reasonable frame-rate and game experience. With a reasonable frame-rate we mean that the user should not notice any frame-rate drops which will lower the gaming experience. We need to list both minimum hardware requirements and software requirements and what versions of software that's needed (Java version etc.).

- We will do our own background music, and we need to take into account what type of mood the music should be composed for.

2.6 Technologies and risks

The game will be developed for the Java-platform because of it's portability and our existent knowledge of the Java language. We're planning to use Lightweight Java Game Library (LWJGL), an open-source library that provides OpenGL and OpenAL-support to Java in a simple and straight forward Application Programming Interface (API). OpenGL is an API for drawing graphics on the screen while OpenAL is an API for processing sound. It's released under the BSD-license model and have reached a stable form. We've chosen to use this library during our game development because Java itself doesn't provide any good hardware acceleration for graphic intensive software.

Our problem here might be that our group has very limited knowledge about OpenGL- and OpenAL-programming, though our Java-experience is at a high level. This can lead to problems during the development stage but we are reasonable sure that we can handle this since we've studied courses that will make our learning progress of this material fast paced.

Another risk could be the closing of LWJGL or Java software that we are using to build the game. This is however very unlikely to happen, because Java is a major language today and is actually opening up their underlying source code as we speak. Since LWJGL is available under the BDS-license it's already out in the open and everyone is free to redistribute the code with the license attached.

OpenAL have recently been widely accepted and is used in many new big commercial games, because it's free and open. It's therefore highly unlikely that they will close access to this software-library in the foreseeable future.

Other risks include changes of product specification and losing a member of the team, but both of them have a low probability occurring.

3 Glossary

2D
Two dimensions.

3D
Three dimensions.

API
Application Programming Interface, an interface that other applications can access to be able to use functions from the software that provides the API.

Functional requirement
This is a feature or function that should be in the system.

Interface
This is the connection into an application or similar to access and control, for example; a graphical user interface has buttons and menus as the interface while a command line interface has text commands as the interface.

Java
Object oriented programming language that compiles and runs in the Java Virtual Machine, also known as the Java Runtime Environment, JRE. It's not open source yet, but is in the process of making it open source. The new version 7 will be completely free and open source under the GNU General Public License Version 2 (GPLv2).

Linux
A free, open-source and UNIX-like operating system.

Linux distribution
A Linux distribution is comprised of a Linux kernel, utilities from the GNU project and a lot of packages (applications). All these packages are normally managed by a packet managing system.

LWJGL
LightWeight Java Game Library, is a free and open-source library that provides an API for developers to use. This contains game, graphics and sound functionality that helps developers to faster and easier develop games in Java. The OpenGL and OpenAL libraries are included in this package.

Mac OS
A UNIX-like operating system from Apple.

Non-functional requirement
Expected behaviour or constraint on the system.

OpenAL
Open Audio Library is a free and open-source library for audio output.

OpenGL
Open Graphics Library is a free and open-source library for graphics output.

Open-source
Software which have its source code available for everyone and which everyone can download and change. Depending on the licence, the source code can be distributed

Operating system
Software that controls the hardware and provides a platform, API, for applications to run on.

Terminal
It's a text-based command line interface where the user can type in commands to do different things like starting a program, in contrast to a graphical user interface where you click on the desired program.

Texture
An image applied to a surface of an object.

Ubuntu Linux
A Linux distribution from Canonical.

UNIX
An operating system from 1969 which was widely used because of its multi-user and multiprogramming abilities.

Windows
A operating system from Microsoft.

4 User requirements definition

4.1 Functional requirements

4.1.1 General

1. **Tutorial text** (*High priority*)
   A tutorial text shall be displayed for the user
   Rationale: A tutorial text helps the user to learn how to play the game. This will also make the user develop his or her skills.

2. **Durance of play** *(High priority)*
   The user shall be able to play the game until the in-game character falls to the bottom of the screen, from the characters perspective.
   Rationale: No limit of game play (falling down) makes the game more challenging and not very predictable.

3. **Screen size** *(Medium priority)*
   The game shall be able to be played in fullscreen or window mode.
   Rationale: Different users tend to have different desires of the screen size and by providing this option to the user we can make those people satisfied.

   UC: Changing screen size

   Primary actor: User
   Stakeholders and interests: User: Wants to change the screen size.
   Preconditions: The game is started.
   Minimal Guarantee: The system is still running and the user is in the menu system.
   Success Guarantee (postconditions): The user was able to change the screen size without any trouble.
   Trigger: The user wants to change the screen size.
   Main Success Scenario (or Basic Flow):

   1. The user selects Options menu alternative options.
   2. The user selects Fullscreen menu alternative and the window will go to fullscreen mode.
   3. The user selects the Back menu alternative and returns to the main menu.

Extensions:

1.

   1. The user selects the High Score menu alternative.
      1. The high score lists is shown.
   2. The user selects the Start game menu alternative.
      1. The game starts.
   3. The user selects the Quit menu alternative.
      1. The game shuts down.

2.

   1. The user selects the Mute menu alternative.
      1. The sound (both in game and menu) turns off.
   2. The user selects the Back menu alternative.
      1. The user returns to the main menu (step 2).

Frequency of occurrence: At least once in a gaming career.

4. **Appearance of sound** *(Low priority)*
   The appearance of sound shall be optional through the menu system.
   Rationale: Different users tend to have different desires of the appearance of sound and by providing this option to the user we can make those people satisfied.

4.1.2 Game

1. **Sideways movement** *(High priority)*
   The user shall be able to move the character to the right and left. When the character stands still and the user moves the character, it has to accelerate before reaching a constant speed. When the character is moving and the user moves character in another direction, it has to de-accelerate before stopping. Thereafter the character can accelerate in the new direction. The user can move the character in mid-air as well as on the ground.

   UC: Moving the character sideways
   Primary actor: User
   Stakeholders and interests: User: Wants to move the character sideways.
   Preconditions: The user is currently playing the game.

Minimal Guarantee: The system is still running.
Success Guarantee (postconditions): The character moves sideways in the desired direction.
Trigger: The user wants to move the character sideways.
Main Success Scenario (or Basic Flow):

1. The character stands still on the ground.
2. The user moves the character sideways.
3. The character accelerates and then reaches maximum velocity.

Extensions:
1.

1. The character is in mid-air.

Frequency of occurrence: Very often.

2. **Character jump** *(High priority)*
The user shall be able to make the character jump. When the character is standing still and the user jumps, the character leaves the ground and reaches a maximum jump height. When the character is moving and the user jumps, the character leaves the ground with the same maximum jump height in the moving direction.

UC: Jumping in-game
Primary actor: User
Stakeholders and interests: User: Wants to make the character jump.
Preconditions: The user is currently playing the game.
Minimal Guarantee: The system is still running.
Success Guarantee (postconditions): The character jumped in the given direction.
Trigger: The user wants to make the character jump.
Main Success Scenario (or Basic Flow):

1. The character stands still on the ground.
2. The user makes the character jump in a given direction.
3. The character lands on a block.

Extensions:
1.

1. The character is moving.

3.

   1. The character misses the block and falls down.
      1. The user gets game over.

   Frequency of occurrence: Very often.

3. **Collect special item** *(Medium priority)*
   When the character touches an item, it disappears from the in-game world and is then ready to be used.

   UC: Picking up a special item
   Primary actor: User
   Stakeholders and interests: User: Wants its character to pick up an item.
   Preconditions: The user is currently playing the game and the user sees an item on the screen.
   Minimal Guarantee: The system is still running.
   Success Guarantee (postconditions): The character picks up the special item.
   Trigger: An item has appeared and the user wants to have it.
   Main Success Scenario (or Basic Flow):

   1. The user makes the character jump towards a block, with an item on it.
   2. The user makes the character land on the block and touch the item.
   3. The item disappears from the in-game world and is ready to be used.

   Extensions: -

   Frequency of occurrence: A couple of times during each game session.

4. **Use special item** *(High priority)*
   When the character is in mid-air and jumps again, a new jump will be made originating from the current position.

   UC: Using a special item
   Primary actor: User
   Stakeholders and interests: User: Wants its character to make a special jump.
   Preconditions: The user is currently playing the game.
   Minimal Guarantee: The system is still running.

Success Guarantee (postconditions): The character makes a special jump.
Trigger: The user wants its character to make a special jump.
Main Success Scenario (or Basic Flow):

1. The user makes the character jump to a block higher up, but realizes that the jump will not be enough to reach the block.
2. The user makes the character jump once more while still in the air.
3. The character gets an extra push upwards and lands on the aimed block.

Extensions:

3.

1. The user doesn't have any special items left to use.
    1. The extra push doesn't happen.

Frequency of occurrence: A couple of times during each game session.

5. **Screen movement** *(High priority)*
   The screen shall move upwards (characters point of view) in a certain speed and the user needs to keep up with the pace or he/she will lose the game.

   Rationale: The screen has to move in order for the player to progress in the game.

6. **Ability to jump through blocks** *(High priority)*
   The character shall be able to jump through the blocks on his way up, but he shall not be able to fall through them on his way down.
   Rationale: The character has to be able to jump through a block underneath them to avoid getting trapped.

7. **Difficulty level** *(High priority)*
   The user shall be able to choose a level of difficulty.

   UC: Starting a game and select difficulty level
   Primary actor: User
   Stakeholders and interests: User: Wants to start a game and play on a level according to the user skills.
   Preconditions: The game is started.
   Minimal Guarantee: The system is still running.

Success Guarantee (postconditions): A game has started and the user was able to select the desired difficult level.
Trigger: The user wants to start a game.
Main Success Scenario (or Basic Flow):

1. The user selects the Start game menu alternative.
2. The user selects the Easy difficulty menu alternative.
3. The user reads the tutorial text which is shown on screen, to learn how to play the game.
4. The user confirms that he/she wants to start the game.
5. The game starts.

Extensions:

2.

1. The user selects the Normal difficulty menu alternative.
2. The user selects the Hard difficulty menu alternative.

Frequency of occurrence: At least once every game session.

8. **Screen speed** *(High priority)*
The screen speed shall increase if a higher difficulty level is chosen.
Rationale: A faster screen-movement will lead to a bigger challenge for the user.

9. **Screen speed increase indication** *(Medium priority)*
There shall be visual and audible indication when the speed level increase.
Rationale: The user must be aware of the speed increment before it happens in order to cope with the speed change.

10. **Receiving points** *(High priority)*
The user will receive certain amount of points for every block he/she passes. The user doesn't need to jump to every block to receive points for them, only pass them in the y-direction.
Rationale: The user needs to something to compare their previous efforts with the recently achieved result.

11. **Showing score** *(High priority)*
The user shall be able to see his current score while playing.
Rationale: The user needs to know about his or her current progress.

12. **Flip function** *(High priority)*
    After a random amount time the screen shall flip and the user needs to continue its journey in the new direction.
    Rationale: A flip function makes the game less predictable and adds a new dimension to the game.

13. **Flip indication** *(High priority)*
    One second before the screen flips an indication of the flip direction will be shown on screen. The screen can only flip 90 degrees at a time.
    Rationale: The user must be aware of the screen flip in order to know which direction to progress in.

14. **Mirroring sideways movement** *(Medium priority)*
    When the in-game character jumps outside of the screen (from his view: left or right), the character will then reappear on the other side of the screen at the same height.

    UC: Moving the character outside the screen
    Primary actor: User
    Stakeholders and interests: User: Wants to move the character to the other side of the screen.
    Preconditions: The user is currently playing the game.
    Minimal Guarantee: The system is still running.
    Success Guarantee (postconditions): The character moves to the other side of the screen.
    Trigger: The user wants to move the character to the other side of the screen..
    Main Success Scenario (or Basic Flow):

    1. The character makes a jump to a side of the screen.
    2. The character disappears.
    3. The character is immediately transported to the other side of the screen.

    Extensions: -

    Frequency of occurrence: Very often.

4.1.3 High score

1. **Storing of high score** *(High priority)*
   The system shall be able to keep track of the high score list so the users can compare the results with their previous efforts (locally stored).
   Rationale: A high score list lets the user compare their results against previous results.

2. **Enter high score list** *(High priority)*
   The user shall be able to enter the high score list when the score is better than one of the existing scores.

   UC: Entering name in high score list
   Primary actor: User
   Stakeholders and interests: User: Wants to enter his/her name on the high score if he/she got enough points and then return to the menu.
   Preconditions: The user has played the game and lost.
   Minimal Guarantee: The user returns to main menu.
   Success Guarantee (postconditions): The user has returned to the main menu and if the user got enough points, he/she has entered his/her name in the High score list.
   Trigger: User gets game over.
   Main Success Scenario (or Basic Flow):

   1. The user beats at least one score on the High score.
   2. The user enter his/her name for the High score list.
   3. The new High score results is displayed.
   4. The user selects the Back menu alternative and returns to the main menu.

   Extensions:
   1.

   1. The user didn't beat any score on the High score.
      1. Jump to stage 3.

   Technology and data variations list: Reads/writes the High score from/to a file.

   Frequency of occurrence: Every game session.

3. **Check high score** *(Medium priority)*
   The user shall be able to check the current locally stored high score list.
   Rationale: An option to view the high score list is important for the serious gamer as well as the beginner to show about their progress and skills without having to start a new game.

4. **Reset high score** *(Low priority)*
   The user shall be able to reset the high score list.
   Rationale: The user might want to reset the high score list if the score is too hard to beat.

5. **High score sound** *(Low priority)*
   The system shall play a special sound effect when the a previous score on the list has been beaten (when the user has lost the game).
   Rationale: A special sound effect will alert the user that he or she is now on the high score list.

4.2 Non-functional requirements

4.2.1 Usability requirements

1. **Tutorial text**

   The user shall be able to learn how to play the game in one minute by reading the tutorial text.

   Rationale: A tutorial text helps the user to learn how to play the game in a faster way than having to find out by themselves.

2. **Easy to install**

   The game shall be easy to install.

   Rationale: An easy to install game is important for not losing any potential user with less computer experience. This is extra important since it's a small game and the patience of the users is lower than with a larger and more complex game.

4.2.2 Performance requirement

1. **Low start up time**

   The game shall at most take five seconds to start after the user has read the tutorial text and starts the game.

   Rationale: Since it's a small game it has less data to load than a typical published game and the user then expects our game to load quick.

4.2.3 Space requirement

1. **Game size**

   The game shall at maximum take 20MB.

   Rationale: The game needs to be small enough to send between friends and distributed over the Internet.

4.2.4 Portability requirement

1. **Multi platform playability**

   It shall work on OSX, Windows and Linux.

   Rationale: To attract as many users as possible at least on OSX, Windows and Linux.

4.3 Standards to be followed

The product and process standards that we must follow are:

- Easy to follow menu system
- Standardized way of moving the character
- Behaves and looks like an original 2D-game

If we have these factors in mind when constructing the game, the users will feel welcome and more inclined to play our game.

5 System architecture

5.1 Overview of system architecture

5.1.1 Graphics

This object only handles the graphical part and do not affect any other part of the system. It will receive information from the Game logic object for processing and then display the result on the screen.

5.1.2 Sound

This object only handles the audio and do not affect any other part of the system. It will receive information from the Game logic object for processing and then send the result to the sound card.

5.1.3 Game logic

This object controls everything. This will contain the main loop of events. It will control everything from the game physics and game rules to what is to be heard and seen (sound and graphics). It will receive information from the Input object about what the user is up to, and acts accordingly. Sends and receives information to and from the File system object.

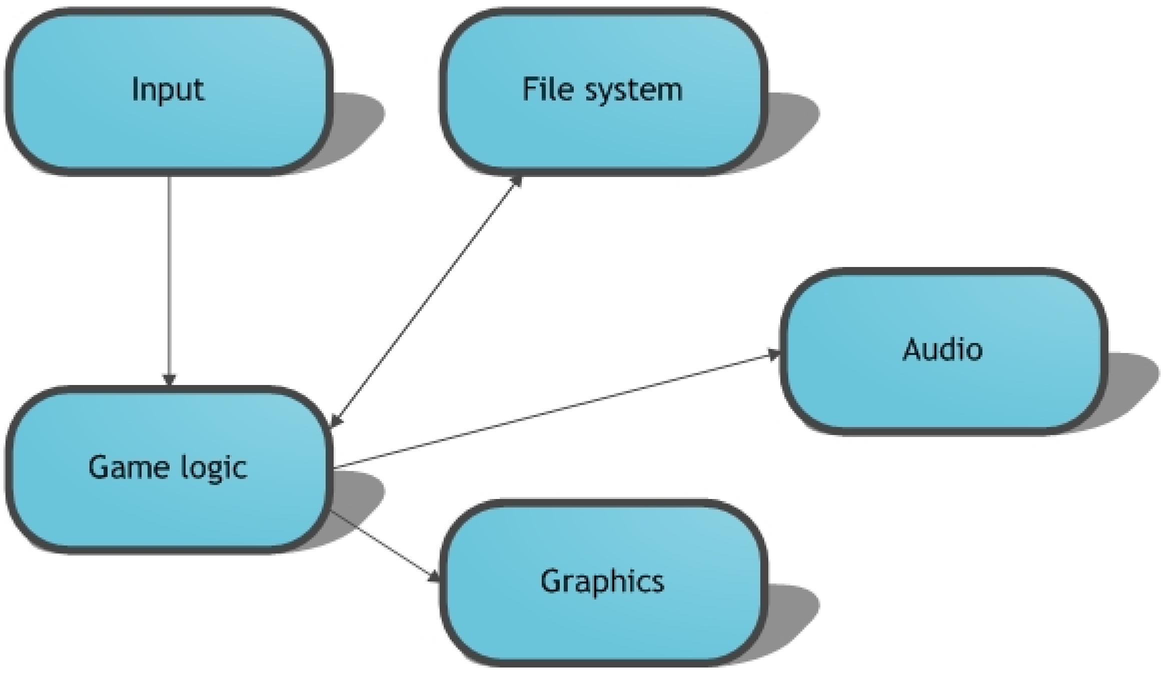
5.1.4 Input

This object interprets the input from the keyboard and sends the information to the Game logic object.

5.1.5 File system

This object controls the input and output from and to files, such as textures, sounds, high score list, settings and similar.

5.2 Illustration of the architectural model



5.3 Reused components

Everything will be reusable since everything we'll write will be modular.

6 System requirements specification

6.1 Functional requirements

6.1.1 General

1. **Tutorial text** (*High priority*)
   Function: Display a tutorial text.
   Description: Displays a tutorial text for the user to learn how to play the game and to make the user develop his or her skills.
   Inputs: None.
   Source: The application.
   Outputs: The tutorial text is shown.
   Destination: The game screen.
   Action: The system handles the users choice of difficulty level. Afterwards the system prints hard coded tutorial text on the gaming screen.
   Pre-condition: User has started a new game and selected a difficulty level.
   Post-condition: A tutorial text has been shown.
   Side effects: None.

2. **Durance of play** *(High priority)*
   Function: Play time.
   Description: Allow the user to play until the character falls to the bottom of the screen.
   Inputs: None.
   Source: The application.
   Outputs: Amount of play time.
   Destination: The game screen.
   Action: The system lets the user play the game until it notices that the users character has fallen to the bottom of the screen. When it happens, the user gets game over.
   Pre-condition: A new game has started.
   Post-condition: The user gets game over.
   Side effects: None.

3. **Screen size** *(Medium priority)*
   Function: Adjusting the screen size.
   Description: Changes the size of the screen to oblige the user preferences.
   Inputs: Chosen screen size.
   Source: The user.
   Outputs: Change in screen size.
   Destination: The game screen.
   Action: The system handles the users choice of screen size and changes

the screen accordingly. If the user selects window mode, the game will be presented in a window drawn on the computer desktop. If the user selects fullscreen mode, the game will take up the entire screen.
Pre-condition: The system has started and the user is in the menu.
Post-condition: The screen size has changed.
Side effects: None.

See user requirement for use case.

4. **Appearance of sound** *(Low priority)*
Function: Option of sound
Description: Changes the appearance of sound in the game, muted or full sound.
Inputs: Chosen sound option.
Source: The user.
Outputs: Sound is played or not played.
Destination: Computer speakers.
Action: The system handles the users choice of the sound option. If the user has selected mute, no sound will be played. If the user has selected full sound, music and in-game sound effects will be played.
Pre-condition: The system has started and the user is in the menu.
Post-condition: Sound is played if chosen.
Side effects: None.

6.1.2 Game

1. **Sideways movement** *(High priority)*
Function: Move sideways.
Description: Moves the character sideways as the user commands.
Inputs: User input.
Source: The user.
Outputs: The character moves sideways.
Destination: The game screen.
Action: The system handles input from the user and moves the character accordingly. If the character stands still, the system will accelerate the character before reaching maximum speed. This will work even if the character is in mid-air.
Pre-condition: A new game has started and user is playing.
Post-condition: The character moved sideways.
Side effects: The character might fall down to the bottom of the screen.

See user requirement for use case.

2. **Character jump** *(High priority)*
   Function: Character jumping.
   Description: The character jumps as the user commands.
   Inputs: User input.
   Source: The user.
   Outputs: The character jumps.
   Destination: The game screen.
   Action: The system handles input from the user and makes the character jump accordingly. If the character stands still it can jump upwards. If the character is running it can jump in the moving direction. The character leaves the ground with the same hard coded jump height. The system has to supervise the characters movement to be able to determine how the jump will affect the character.
   Pre-condition: A new game has started and the character is on a block.
   Post-condition: The character has jumped.
   Side effects: The character might fall down to the bottom of the screen.

   See user requirement for use case.

3. **Collect special item** *(Medium priority)*
   Function: Pick up item.
   Description: The character picks up a special item.
   Inputs: User input.
   Source: The user.
   Outputs: The character picks up an item.
   Destination: The game screen.
   Action: The system presents an item on a block on the screen. If the user touches the item, the system has to check if there is a free spot in the characters inventory. If there is a free spot, the item disappears from the in-game world and is ready to be used. If there is no empty place in the inventory list, the item will remain in the world.
   Pre-condition: A new game has started and the user is playing.
   Post-condition: There is at least one item ready to be used.
   Side effects: None.

   See user requirement for use case.

4. **Use special item** *(Medium priority)*
   Function: Use item.
   Description: The character uses a special item and gets a special jump.
   Inputs: User input.
   Source: The user.

Outputs: The character makes a special jump.
Destination: The game screen.
Action: The system handles input from the user to make a special jump while the character is in the air. The system checks if the character has an special item in its inventory list. If it has a new jump will be made originating from the current position. If there was no item, nothing will happen.
Pre-condition: The character is in mid-air.
Post-condition: The character made a special jump if the character had a special item.
Side effects: The character might fall down to the bottom of the screen.

See user requirement for use case.

5. **Screen movement** *(High priority)*
Function: Move the screen.
Description: Moves the screen in a certain speed upwards, pushing the character forward.
Inputs: None.
Source: The application.
Outputs: The screen is moving.
Destination: The game screen.
Action: The system moves the screen in a certain speed depending on the current difficulty level.
Pre-condition: A new game has started and the user is currently playing.
Post-condition: The screen has moved.
Side effects: None.

6. **Ability to jump through blocks** *(High priority)*
Function: Jump through blocks.
Description: The character jumps through blocks on it's way up.
Inputs: None.
Source: The application.
Outputs: None.
Destination: The game screen.
Action: The system allows the character to pass through blocks on it's way up. If the character lands on a block, the system will not let it pass through.
Pre-condition: A new game has started and the user is currently playing.
Post-condition: The character has passed a block.
Side effects: None.

7. **Difficulty level** *(High priority)*
   Function: Choose difficulty level.
   Description: The user selects difficulty level for the game.
   Inputs: Chosen difficulty level.
   Source: The user.
   Outputs: The corresponding screen speed to the selected difficulty level.
   Destination: The game screen.
   Action: The system handles the user's choice of difficulty level and thereafter calculates the proper game screen speed and then sends it forward.
   Pre-condition: The user has selected the start new game option in the main menu.
   Post-condition: Adjust the screen speed accordingly.
   Side effects: None.

   See user requirement for use case.

8. **Screen speed** *(High priority)*
   Function: Speed increment.
   Description: The screen speed increases to higher rate.
   Inputs: Current speed level.
   Source: The application.
   Outputs: New speed level.
   Destination: The game screen.
   Action: The system checks the current speed level and the elapsed time since the last speed increment. If the elapsed time is high enough, the speed will change to a higher level.
   Pre-condition: A new game has started and the user is currently playing.
   Post-condition: The screen speed has changed.
   Side effects: None.

9. **Screen speed increase indication** *(Medium priority)*
   Function: Indication of speed change.
   Description: Indicates the user that the screen speed has changed to a higher rate.
   Inputs: Speed increment.
   Source: The application.
   Outputs: Visual and audible indication.
   Destination: The game screen and computer speakers.
   Action: The system notices a screen speed change and presents a visual and audible indication to the user that the screen speed has changed.
   Pre-condition: The screen speed has increased.
   Post-condition: An indication has been occurred.
   Side effects: None.

10. **Receiving points** *(High priority)*
    Function: Receive points.
    Description: The user receives points for every block passed.
    Inputs: User input.
    Source: The user.
    Outputs: The application.
    Destination: The game screen.
    Action: The system checks if the user has passed a block. If a block has been passed the collected amount of points will increase with the score based on the current speed level.
    Pre-condition: A new game has started and user is currently playing.
    Post-condition: A set of points is received.
    Side effects: None.

11. **Showing score** *(High priority)*
    Function: Show current score.
    Description: Displays the user's current score in-game.
    Inputs: None.
    Source: The application.
    Outputs: A presented score.
    Destination: The game screen.
    Action: The system presents the current users score while playing the game.
    Pre-condition: A new game has started and the user is currently playing.
    Post-condition: The score is shown.
    Side effects: None.

12. **Flip function** *(High priority)*
    Function: Flip the screen.
    Description: The screen flips (rotates) in a certain direction.
    Inputs: None.
    Source: The application.
    Outputs: Rotated screen.
    Destination: The game screen.
    Action: The system will at a randomly selected event rotate the screen 90 degrees to the left or to the right. The game will continue as before in the new direction.
    Pre-condition: A new game has been started and the user is currently playing.
    Post-condition: The screen has rotated.
    Side effects: None.

13. **Flip indication** *(High priority)*
    Function: Indication of rotate direction.
    Description: Indicates the flip direction that the screen will rotate too.
    Inputs: Flip rotation.
    Source: The application.
    Outputs: Flip indication.
    Destination: The game screen.
    Action: The system will present an indication of what direction the flip will have.
    Pre-condition: A flip rotation will occur.
    Post-condition: An indication was shown to the user.
    Side effects: None.

14. **Mirroring sideways movement** *(Medium priority)*
    Function: Mirror movement.
    Description: Moves the character from one side to the other when crossing the side of the screen.
    Inputs: User input.
    Source: The user.
    Outputs: A mirror movement.
    Destination: The game screen.
    Action: The system checks if the character has gone outside the left or the right sides of the screen. If the character is outside, it will be transported to the other side of the screen.
    Pre-condition: A new game has started and the user is currently playing.
    Post-condition: A mirror-movement has happened.
    Side effects: The character might fall down to the bottom of the screen.

    See user requirement for use case.

6.1.3 High score

1. **Storing of high score** *(High priority)*
   Function: Store the high score list.
   Description: Stores the high score list to the local computers hard drive.
   Inputs: User input.
   Source: The user.
   Outputs: Store high score list.
   Destination: Computer hard drive.
   Action: When one score on the high score list has been beaten and the user has entered his or her name, the new high score list will be saved to the local computers hard drive.

Pre-condition: High score has been beaten.
Post-condition: The high score was saved to the hard drive.
Side effects: None.

2. **Enter high score list** *(High priority)*
   Function: Enter the high score.
   Description: Checks the high score list and lets the user enter a name if a score was beaten.
   Inputs: A high score list.
   Source: The computers local hard drive.
   Outputs: User name.
   Destination: The game screen.
   Action: When a game has ended, the system will load the current high score list. If a score on the high score list was beaten, the system will present an option to the user to enter a name.
   Pre-condition: A game has ended.
   Post-condition: A name has been entered.
   Side effects: None.

   See user requirement for use case.

3. **Check high score** *(Medium priority)*
   Function: Check the high score list.
   Description: Presents the current locally stored high score list to the user.
   Inputs: High score list.
   Source: The computer hard drive.
   Outputs: The current high score list.
   Destination: The game screen.
   Action: The system loads the high score list and prints it to the screen for the user to view.
   Pre-condition: The system has started and the user is in the menu.
   Post-condition: The high score was shown.
   Side effects: None.

4. **Reset high score** *(Low priority)*
   Function: Reset high score list.
   Description: Resets all the scores on the high score list.
   Inputs: High score list.
   Source: The computer hard drive.
   Outputs: The reset high score list.
   Destination: The computer hard drive.

Action: The system loads the high score list and resets each value in it.
Pre-condition: The system has started and the user is in the menu.
Post-condition: The high score list was reset.
Side effects: None.

5. **High score sound** *(Low priority)*
   Function: Play high score sound.
   Description: Plays a sound effect when a previous score on the high score list was beaten.
   Inputs: High score list.
   Source: The computer hard drive.
   Outputs: Special sound effect.
   Destination: The computer speakers.
   Action: When a game is over, the system has loaded the high score list and the score is good enough to enter the high score list, the system will play a special sound effect to alert the user.
   Pre-condition: A game has ended and a score on the high score list has been beaten.
   Post-condition: A special sound effect was played.
   Side effects: None.

6.2 Non-functional requirements

6.2.1 Usability requirements

1. **Tutorial text**

   The user shall be able to learn how to play the game in one minute by reading the tutorial text. This text will appear right you when start playing the game and the user will be able to read the text as long as he/she wants. This will contain instructions on how to move the character and game rules.

   Rationale: A tutorial text helps the user to learn how to play the game in a faster way than having to find out by themselves.

2. **Easy to install**

   The game shall be easy to install. It shall only require a double click on the executable file for the game.

   Rationale: An easy to install game is important for not losing any potential user with less computer experience. This is extra important since it's a small game and the patience of the users is lower than with a larger and more complex game.

6.2.2 Performance requirement

1. **Low start up time**

   The game shall at most take five seconds to start after the user has read the tutorial text and started the game. We must make sure that textures and sound files can be loaded as fast as possible.

   Rationale: Since it's a small game it has less data to load than a typical published game and the user then expects our game to load quick.

6.2.3 Space requirement

1. **Game size**

   The game shall at maximum take 20MB. We have to make sure that the size of the textures and audio files are not to large.

   Rationale: The game needs to be small enough to send between friends and distributed over the Internet.

6.2.4 Portability requirement

1. **Multi platform playability**

   The game shall be able to work on a set of different platforms due to Java's platform independence. It shall at least work on OSX, Windows and Linux.

   Rationale: To attract as many users as possible at least on OSX, Windows and Linux.

7 System evolution

7.1 Fundamental assumptions

- Working computer

  The user must have a working computer to start and play the game.

- Keyboard

  The user must have a keyboard connected to the computer to be able to play our game.

- Operating system

  The user has one of these operating systems: Windows XP, Mac OS X and Linux.

- Java >= 1.5

  Since we will write the game for Java version 1.5 the user must have version 1.5 or later of the Java Runtime Environment (JRE).

- Know how to start an application

  The user needs to know how to start the application, which is done either by clicking on the game binary or running the game from a terminal.

- Understand a simple menu system

  The user needs to know how you usually control a normal menu system.

- Understand English

  The user needs to be able to read and understand English, since this game will only feature English text.

7.2 Anticipated changes

7.2.1 Due to hardware evolution

Our game is so small that the hardware evolution, in a performance viewpoint, will not have any effect on it which means that we will not need to change the game because of the hardware evolution.

If hardware vendors decides to drop the support for OpenGL or OpenAL we will have to code around that and use some other libraries. If that happens, there will probably be layers that will handle those calls since OpenGL and OpenAL are quite widespread. Otherwise we can use the Java libraries for graphics and audio.

7.2.2 Changing user needs

It's very unlikely that the user needs will change in a way that there will not be any users left that wants to play a game like ours.

7.2.3 Due to software evolution

If Java decides to drop the backward compatibility we will have to modify the code, if any, where they have made changes.

8 Appendices

8.1 Minimal system configuration

In order to play the game, the user needs to have a computer that is able to run JRE 1.5 or higher. The user also needs to have a keyboard and a video card with OpenGL acceleration. This will give the user a adequate gaming experience. If any of these are missing, the game will not be playable.

8.2 Optimal system configuration

Same as the minimal system configuration, but for a greater experience, the user will also need a sound card. This will enable the user to get the ultimate Flip Jump experience.

9 Index