Requirements and Analysis Document for “Roung Out”

Version: alpha 1.0

Date : 24 /03/17

Main author: Alex Nitsche
Co-author: Adam Grandén

1. Introduction

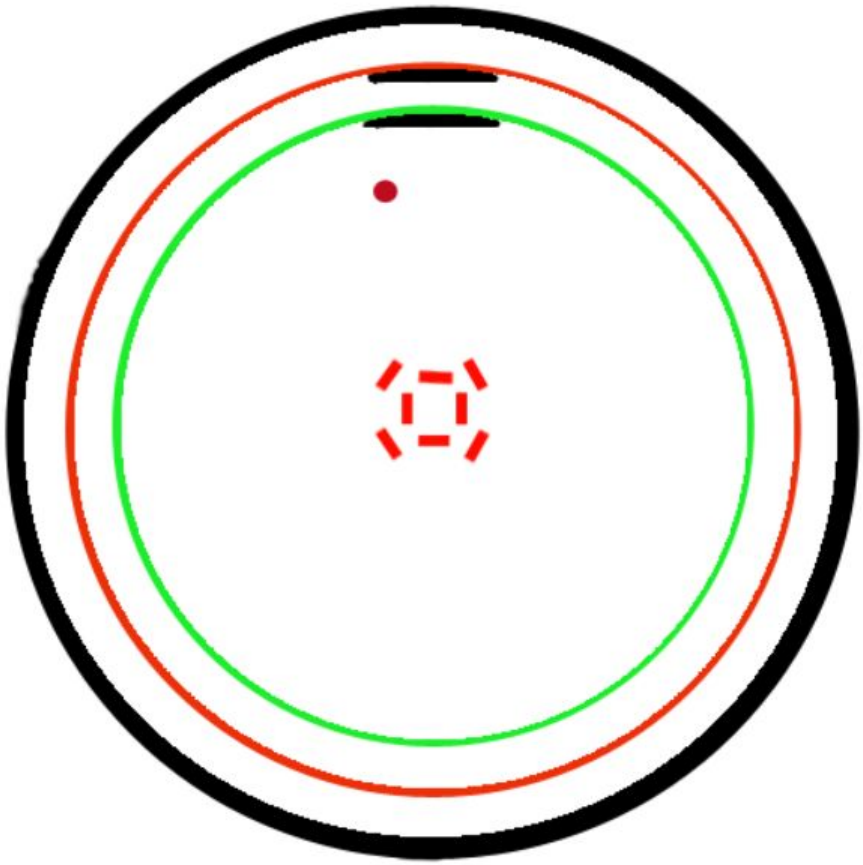
1.1 Background
Our application is a game that we call “Roung Out”, its our modern mix of the two classics “Pong” and “Breakout”. The main *game-loop* is to have two *players*, each controlling a *pad* that is used to bounce a ball in a circular playing area. The ball can hit *bricks* which may contain *power-ups* which is used to make the game more exciting for the *players*. The goal of the game is to gather points. Points are given out when a *player* manages to prevent the opposing *player* to rebound the ball, the ball hits a *brick* or when some specific power-up is collected. Visual and sound feedback for interactions within the game are planned to be implemented.

We think that this mix is needed since we still can add new design, game-elements and complexity to old classics by combining the two games (and add a bit of our own ideas to it). We believe that there is a problem in the *indie-games* genre and that is the abundance of games that simply takes old ideas and put new graphics on top of it instead of trying to innovate and improve on these ideas and we hope that our work will help solving this problem. This game is aimed to children in the age-group 8-12 years with the goal of bringing a moment of joy and happiness to their everyday life. The game is to be used mainly on a desktop environment with the possible transition to a mobile platform.

1.2 Word List
Please look in the separate document “Word Book” that we have made, it lists all our “custom”-words and their respective meanings.

2 Requirements

2.1 User interface Sketches, drawings and explanations of the application user interface (possible navigation).



The outer black circle is the wall of the game, if the ball (the red circle) hits the wall it's reset and points are distributed to the players accordingly. Ther red and green inner circles represent the orbit of the pads (the pads are the black semi-oval shapes), the respective pad is confined to its orbit. When the pad in the green orbit hits the ball, the pads switches orbit, this is also the way we show whose turn it is to hit the ball.

2.2 Functional requirements

A player should be able to start and exit from the application (obviously). A player should be able to start a game. A game should be able to be won by a player, the game must end at some point. While in a game, a player should be able to control their *pad*. With their pad, a player should be able to hit and miss a ball, the ball should be able to hit bricks. A player should also be able to use *power-ups* during the game. The player should be able to change options and go to the main menu at any time.

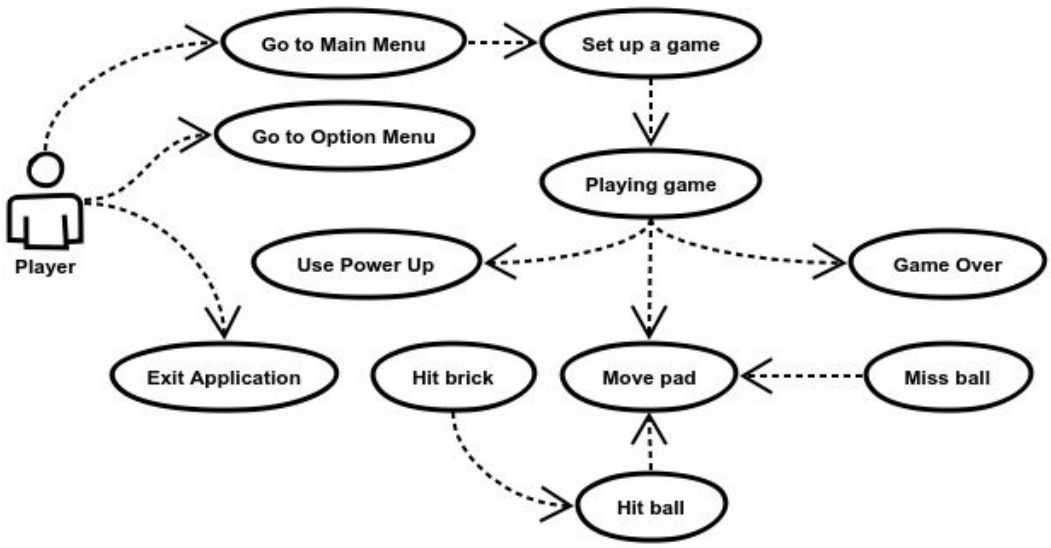
User Cases can be found in *Appendix A*

2.3 Non-functional requirements Any special considerations besides functionality? Usability, reliability, performance, supportability, legal, implementation, … NOTE: Testability mandatory (must have tests)

8/10 users should intuitively be able to start the application and play with little instructions from an experienced user, developers or the game itself.
9/10 users should able to play the game without having any major issues with the readability of the game.
The application should only be allowed to 1 unaccounted termination (crash) in every 100 uses (1% crash tolerance). The application should pass all of our unit-tests with at least 90% code coverage (of our code, code from external libraries are not included in this requirement).
No obligatory support to the application post-release is promised, tough there might be optional post-development.

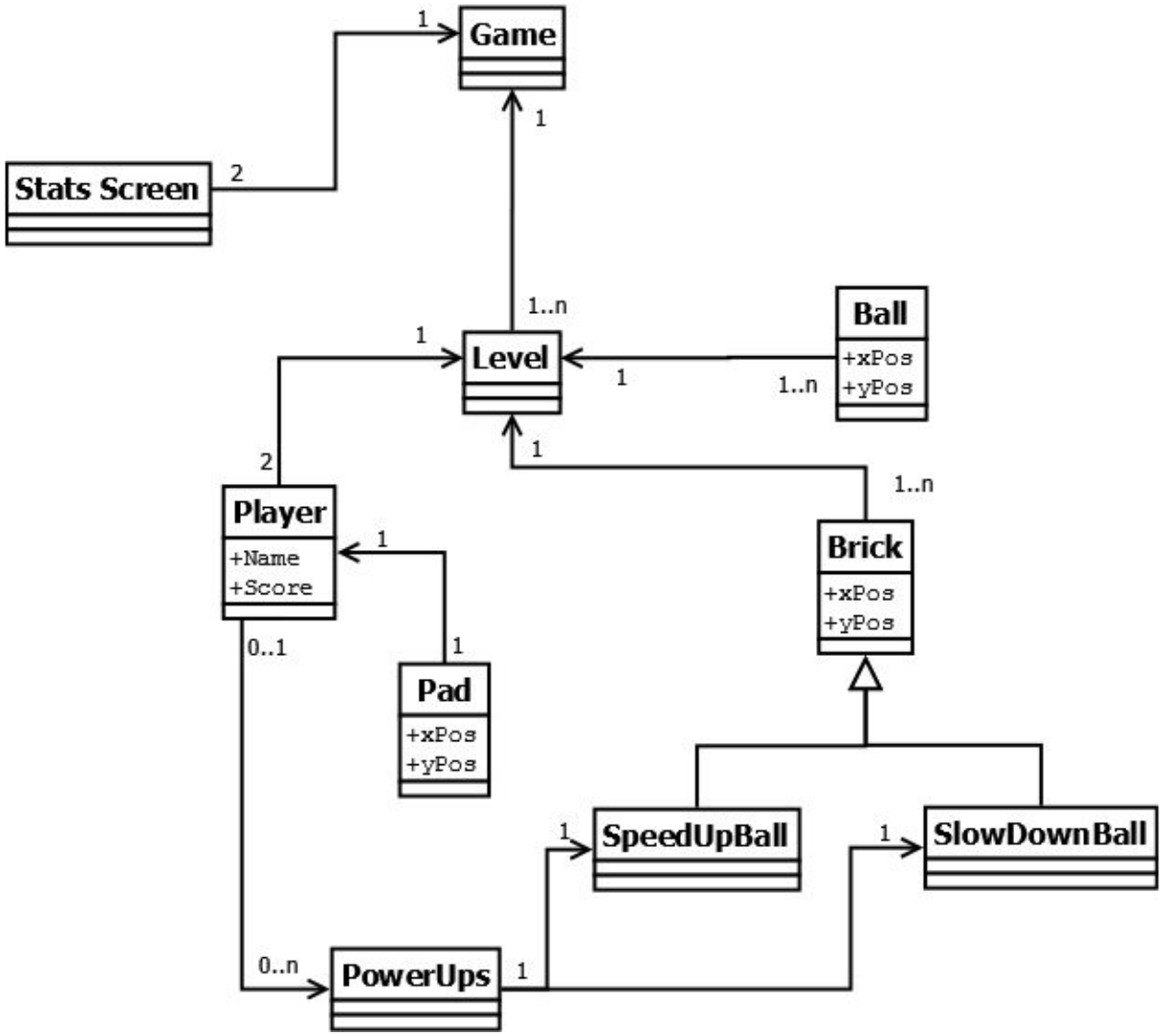
3 Use cases

An UML use case diagram
3.1 Use case listing Use case texts (using the use case template)



4 Domain model

An UML class diagram.
4.1 Class responsibilities Explanation of responsibilities of classes in diagram



5 References