Requirements and Analysis Document for the Infinity Run project (RAD)

Version 1.3

This version overrides all previous versions.

2017-05-28

Authors:

Jacob Rohdin,

Anders Bäckelie,

Leo Oinonen,

Mikael Törnqvist

1 Introduction

The project aim is to make a game where the goal is to travel as far as possible.. When moving the player will be facing different hazards such as spikes and even moving AI. The player will have basic movement such as moving left and right and jumping. The game will increase in difficulty as you progress further away from the start, as the enemies become stronger. However, the player will not be helpless as the character will also be armed with a weapon.

As mentioned earlier the monster will become stronger as you traverse more rooms. This is a scaled difficulty which means spikes will deal more damage if the players has cleared many rooms and possibly kill you instantly depending on the characters max HP. The character will among other things have HP, attack damage and a movement speed.

Coins, which is this game's currency can be collected during as the player travel through the world. These coins may be used to purchase upgrades, the more upgrades the “stronger” or more advanced the character will become. When the character dies, they will return to a shop screen where the player can purchase upgrades for their character with the collected coins. these upgrades consist of Increasing health, speed or the number of available jumps.

The app will be a desktop, singleplayer game for the platforms Windows/Mac/Linux and will be using a simple graphical user interface.

General characteristics

- The application will be a single-player 2D platformer-style game, meaning that the player character will have to move in two dimensions (up/down; left/right) in order to traverse a world filled with platforms.
- The world will be procedurally generated using pre-defined building blocks designed to fit together in order to make an “infinitely” large world. The point of origin will always be the same.
- The application will keep track of the player’s “score”. The score is defined as the number of rooms you have traversed from the centerpoint.
- In addition to the score, the player will be able to collect coins generated onto the world. These coins will persist through several game sessions, i.e. they are not bound to the current session.
- The game session will end when the player’s health points reach zero, referred to as “death”. This occurs by taking damage, and is caused by various hazards (such as spikes or mobile enemies) generated onto the world.
- Upon dying, the player will have access to the shop. In the shop the player will be able to upgrade their character using the coins collected during the game sessions.
- The application will be using a GUI that will include the world and all its constituents, a coin counter, and a health counter. The shop will have a number of upgrades to choose from, their price, and the coin counter.

1.1 Definitions, acronyms, and abbreviations

Technical definitions

- **GUI** - graphical user interface.
- **Java** - platform independent programming language
- **JRE** - the Java Runtime Environment. Additional software needed to run an Java application.
- **AI -** Artificial Intelligence or in other words bots that are programmed to exhibit situation-dependent behaviour.
- **MVC** - a program modeling pattern to separate the view, controller and model. The point is that you can replace one of these three parts without affecting the other two parts.

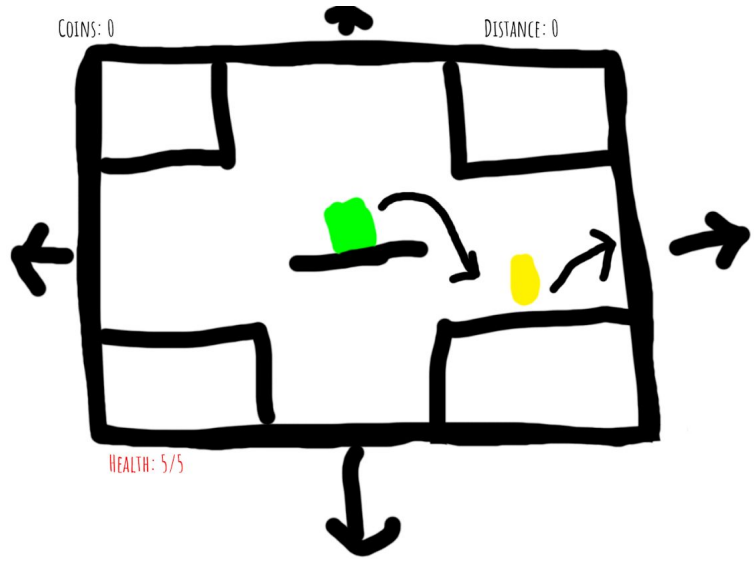
Game Definitions

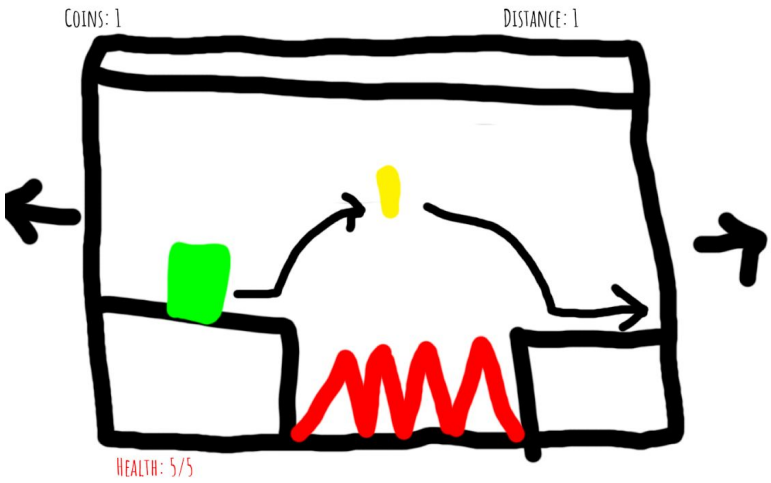
- **HP** or Health Points, the total number of damage something can take before dying.
- **Movement Speed** is the speed of which things can move.
- **Coins** which is the currency of the game, used for purchasing upgrades and similar things. The number of coins looted can also be increased with upgrades.
- **Looting** determines how many coins you will get per coin object you pick up.
- **Rooms** are the created rooms that the game generates the world with, the world has many rooms which are randomly determined by an algorithm.
- **Death** when your HP reaches zero, you will then be transported to the shop. This only means you have to start from the starting point again - your character will not be removed.
- **Shop** is the place where you can spend your coins for upgrades or weapons.
- **Upgrades** will be making your character better through more HP, more jumps, higher jumps, higher damage, more critical hit chance, critical hit damage and better looting.
- **Score** will be measured in number of rooms cleared.
- **Damage** is the number of HP something can deal in one hit.
- **Critical Hit Chance** is the chance you have to do a special attack that will deal increased damage depending on your *critical hit damage*.
- **Critical Hit Damage** increases the damage of your critical hits. Without any critical hit damage upgrades you deal 2 times your normal damage.
- **Monsters** will have some sort of AI, they can be killed if you have enough damage or outrun with enough movement speed or jumping “power”..
- **Character** is the one you as a player control, you can make the character stronger by collecting coins which later are used for purchasing upgrades.
- **World** is an infinite puzzle of rooms, each room’s entrance is put together with another room’s entrance which makes for a never ending world with unlimited exploration.
- **Air Jump** is a jump performed in the air, you will be able to do double jumps and even more jumps depending on your upgrades.

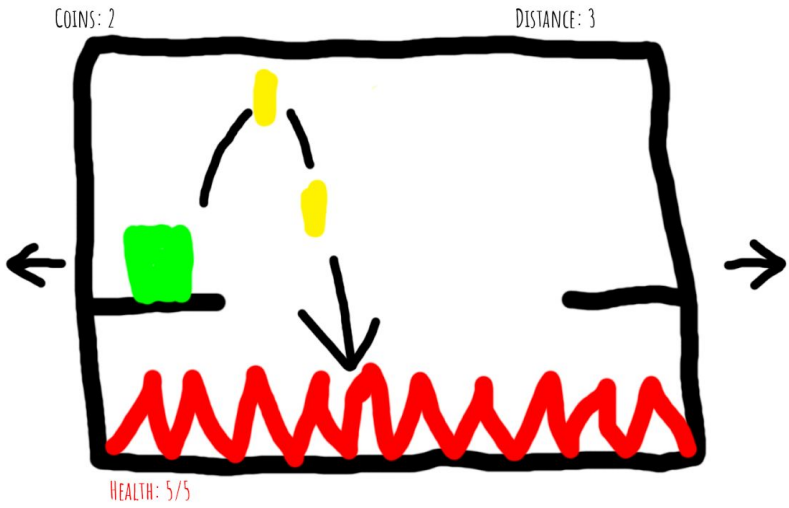
2. Requirements

2.1 Graphical User Interface

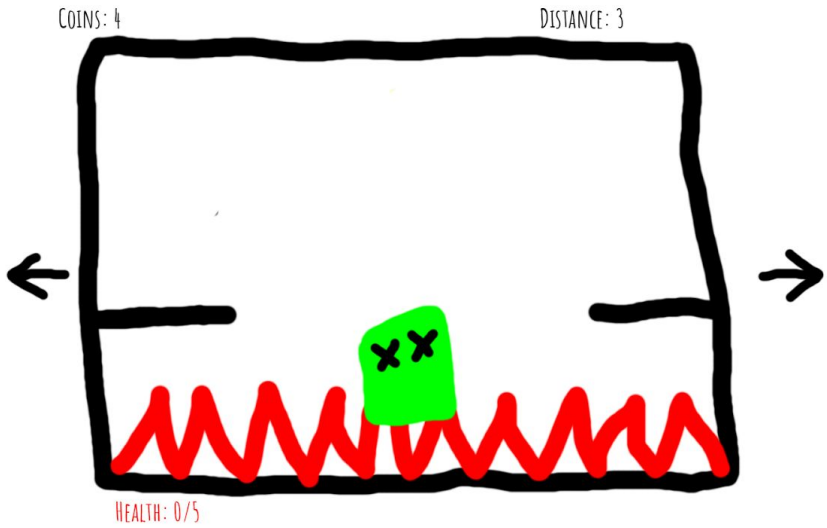
The game GUI will consist of a character, coins, hazards, coin counter, distance counter and health. In the sketches below, the player is green, hazards are red, and coins are yellow.





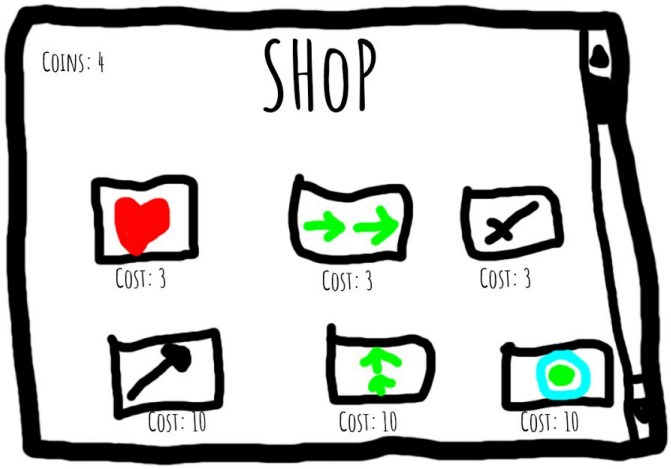


The player jumps for the coins but misses the platform on the other side, they hit the damaging obstacle and lose enough of the HP To Die.



As the player died, they are taken to the shop. Below is a simple sketch of what the shop

might look like:



2.2 Functional Requirements

1. Start the game at the beginning area
2. Traverse the world
    a. The player can move (walk right/left and jump)
    b. The player can collect coins
    c. The player can attack monsters
    d. The player will be able to get hurt by monsters or other hazards
    e. Generate surrounding rooms when entering a room
    f. Score points based on rooms cleared
3. Death upon losing all HP
4. Purchase upgrades for the coins collected during the traversing of the world

2.3 Non-functional requirements

2.3.1 Game

The game should include in-game instructions in order to make it easier for new players to start playing.

The game should be fun and should be easy to play. This means you should only have little or none customization for your character before you actually can start the game and play.

2.3.2 Technical

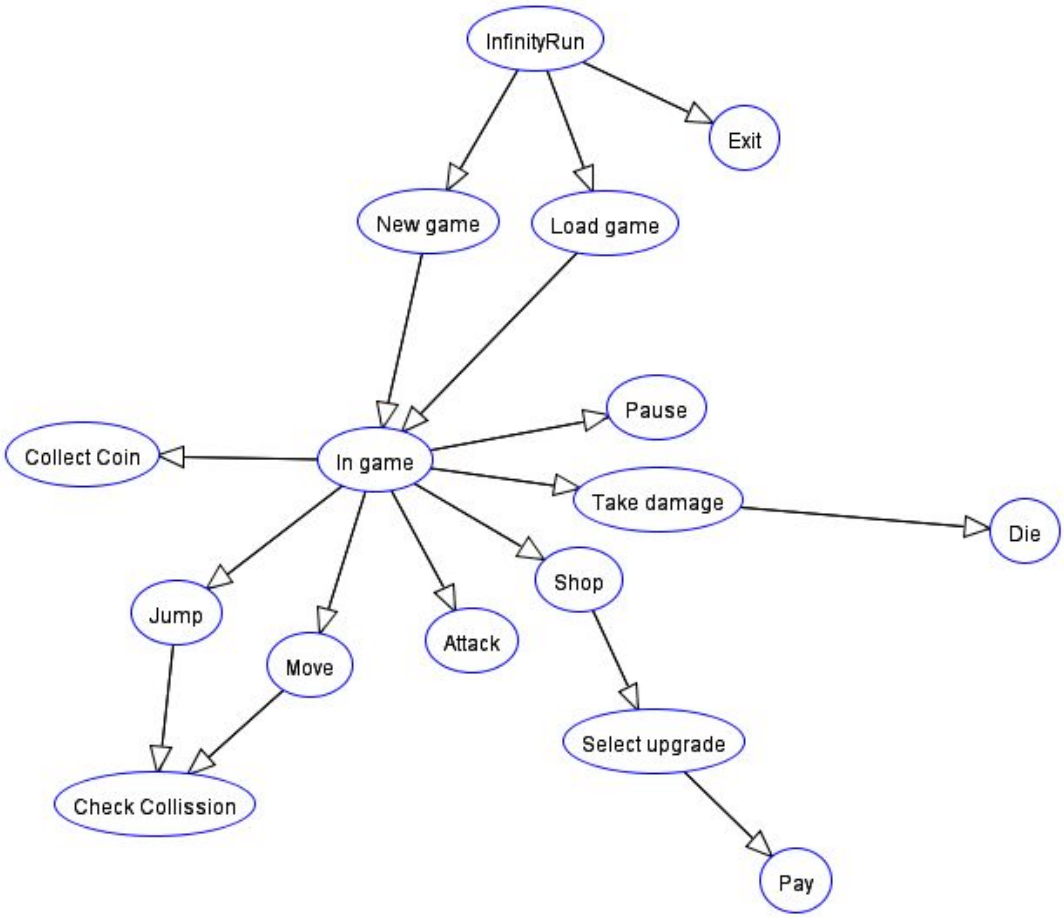
The application should be designed so that it can use different graphics libraries/frameworks, i.e no part of the framework should be found in the classes responsible for the model.

The game should be well documented so that future improvements/changes can be made with less effort.

The GUI should be easy to move to a mobile platform like android.

3. Use cases

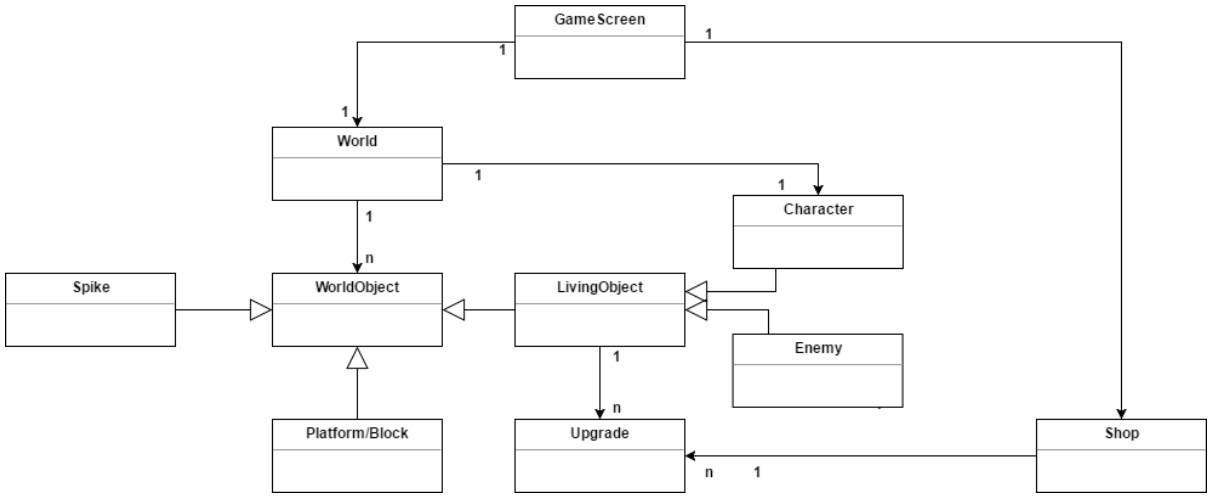
Use Cases Overview



Use cases

See Appendix A

4. Domain Model



4.1 Class Responsibilities

InfinityRun is the main class we have when we open the game. This leads to a screen where the player can either start a game with a new character or open another screen to load previous characters. Either of these options instantiates a world with a character to start the game.

The World class is a object container. It will have a list of all the objects in the world, so we can display them properly for the players and some objects may just be used for collisions.

WorldObject is the “base” class in the visible objects hierarchy. It will have some basic functions like Position and methods to get the information that is required to render the object.

LivingObject extends WorldObject and contains the different methods that enables it to move and accept different upgrades, depending on what kind of LivingObject it is. It’s primary function is a template for intelligent WorldObjects.

Hazard extends WorldObject and acts similar to it but a Hazard is supposed to be a baseclass for spikes or similar, something that can damage the character. Objects like this is better treated separately because we will make different things if the player intersect with one of these objects.

Enemy extends LivingObject. The main reason that enemy is not represented as a hazard is because the enemy is a living object with upgrades like the character. The AI of the Enemy is implemented in the frame method.

Platform extends WorldObject, Platform is just a simple object in the world, the main difference between the platform and the standard worldObject is that the platform has a set texture and sizes that fit.

Character extends LivingObject. The character class will represent the actual player in the world. The Characters movement will be controlled by the keyboard. The character class will also have a List with upgrades that this Character has previously bought.
The player will load in this character into the world though the LoadScreen class.

Shop is a class where the “user” can buy upgrades the active Character. When you purchase an upgrade it increases the level of the upgrade you picked by one and then removes coins from your character.

WoWrapper is the Wrapper class for worldobjects using the util functions distance and getcenter. This is to ensure that the utils function use generic components and not Project-dependent objects.

HUD is a class that displays health and coins derived from the current instance of the character wrapped in a package.

Upgrade is a class that contains generic information about upgrades, and the methods to add levels to a character which the shop implements. This information is also used by the specific upgrades such as; health, speed, jumpH, criticalHitChance, criticalHitDamage and so on.

MeleeWeapon is a class that extends WorldObject and creates an abstract template of one of the multiple melee weapons that exist. This template is used by the specific weapons in the weapons package. This weapon is implemented by a livingObject, and enables them to attack other LivingObjects within the weapons range. It also takes in values from some of the upgrades such as critical hit chance and damage along with melee handling.

SaveCharacter is a class that saves the current values of the upgrades of the active character to a textfile(savefile). The number of coins that the Character currently has will also be saved to the same file.

LoadCharacter is a class that reads data from the savefile and loads the data as the selected character.

Appendix A

InfinityRun

Summary: A startmenu with three buttons, one to create a new character, one to load a former character and one to exit the game.
Priority: High
Extends: None
Includes: New Game, Load Game, Exit Game
Participators: User

New game

Summary: Starts the game with an empty character
Priority: High
Extends: InfinityRun
Includes: In-game
Participators: User

	Actor	**System**
1	Left mouse click on “New Character” button	-
2	-	System switches to gamescreen, creates new world and new character
		Go to In-Game

Load game

Summary: Starts the game with an existing character
Priority: High
Extends: InfinityRun
Includes: In-game
Participators: User

	Actor	**System**
1	Left mouse click on “Load Character” button	-
2	-	Opens loadscreen with buttons corresponding to Saved Characters
3	Presses button “Save #”	-
4	-	System switches to gamescreen, creates new

		world and calls the loadcharacter class
5	-	LoadCharacter loads character data from txt file and gives it to world

Exit

Summary: Leftclicks exitbutton and exits the game.
Priority: Low
Extends: InfinityRun
Includes: None
Participators: User

	Actor	**System**
1	Left mouse click on “Exit” button	-
2	-	Exits the game.

In-Game

Summary: The main state of the game. The game’s loop.
Priority: High
Extends: New Game, Load Game
Includes: Collect Coin, Jump, Move, Attack, Shop, Take Damage, Pause
Participators: User

Attack

Summary: The user presses attackbutton (“space”) and the game checks if the attack hit something, if the attack is a critical hit and if the object attacked survives.
Priority: High
Extends: In-game
Includes: None
Participators: Enemy, Character
Normal flow of events: Attack is on cooldown and no attack is made.

	Actor	**System**
1	Presses Spacebar	-
2	-	Check if attack is on cooldown

3	-	Attack is on cooldown
4	-	No attack is made.

Alternative flow: Attackable objects is out of range, attack misses.

	Actor	**System**
1	Presses Spacebar	-
2	-	Check if attack is on cooldown
3	-	Attack isn't on cooldown
4	-	Check if attackable object is within range
5	-	Attackable object is not within range
6	-	Attack misses

Alternative flow: Attackable objects is within range but no critical hit.

	Actor	**System**
1	Presses Spacebar	-
2	-	Check if attack is on cooldown
3	-	Attack isn't on cooldown
4	-	Check if attackable object is within range
5	-	Attackable object is within range
6	-	Check if the attacks is a critical hit.
7	-	Attack isn't a critical hit
8	-	Calculates new health of enemy and gives a knockback.

Alternative flow: Attackable objects is within range and does a critical hit.

	Actor	**System**
1	Presses Spacebar	-
2	-	Check if attack is on cooldown
3	-	Attack isn't on cooldown
4	-	Check if attackable object is within range
5	-	Attackable object is within range
6	-	Check if the attacks is a critical hit.
7	-	Attack is a critical hit
8	-	Calculates new health of enemy and gives a knockback.

Alternative flow: Attackable objects is within range and does a critical hit and the enemy dies.

	Actor	**System**
1	Presses Spacebar	-
2	-	Check if attack is on cooldown
3	-	Attack isn't on cooldown
4	-	Check if attackable object is within range
5	-	Attackable object is within range
6	-	Check if the attacks is a critical hit.
7	-	Attack is a critical hit
8	-	Calculates new health of attackable object.Health is less than 0.

9	-	The object despawns.

Pause Game

Summary: The user presses pausebutton (“Esc”) and the game is paused.
Priority: Medium
Extends: In-game
Includes: Mainmenu, Exit Game, Unpause
Participators: User
Normal flow of events: Pauses the game.

	Actor	**System**
1	Presses the key “Esc”	-
2	-	Pauses the game + opens “Pause View”

Alternative flow: Unpauses the game.

	Actor	**System**
1	Presses the key “Esc” or leftclicks the “unpause” button	-
2	-	Unpauses the game + exits “Pause View”

Exit

Summary: The user leftclicks exit and saves the game.
Priority: Low
Extends: Pause Game
Includes: None
Participators: User

	Actor	**System**
1	Left mouse click on “Exit” button	-
2	-	Exits and saves the game

Mainmenu

Summary: The user leftclicks mainmenu button and goes back to the mainmenu when the game progress is saved.
Priority: Low

Extends: Pause Game
Includes:
Participators: User

	Actor	**System**
1	Left mouse click on “Mainmenu” button	-
2	-	Moves to mainmenu and saves the game

Jump

Summary: The participator presses the jumpbutton “arrowkey up” and performs a jump.
Priority: High
Extends: In-Game
Includes: None
Participators: Character, Enemy
Normal flow of events: The participator tries to jump but is in air without double jump charges.

	Actor	**System**
1	Presses the ”Up Arrowkey”	-
2	-	Check so the participator has doublejump charges
3	-	No charges
4	-	Check so the participator is on the ground
5	-	Participator is in the air
6	-	No Jump

Alternative flow: The participator has charges and tries to jump but collision stops the participator..

	Actor	**System**
1	Presses the ”Up Arrowkey”	-
2	-	Check so the participator has doublejump charges
3	-	Got charges left.

4	-	Check so the participator is on the ground
5	-	Participator is in air.
6	-	Check collision
7	-	Collision in the jump direction
8	-	Stops movement in y-direction and removes charge

Alternative flow: The participator has no charges but is on the ground. Collision stops the jump.

	Actor	**System**
1	Presses the ”Up Arrowkey”	-
2	-	Check so the participator has doublejump charges
3	-	No charges
4	-	Check so the participator is on the ground
5	-	Participator is on the ground
6	-	Check collision
7	-	Collision in the jump direction
8	-	Stops movement in y-direction

Alternative flow: The participator has charges and is in air. Does jump.

	Actor	**System**
1	Presses the ”Up Arrowkey”	-
2	-	Check so the participator has doublejump charges
3	-	Got charges left.

4	-	Check so the participator is on the ground
5		In air
6	-	Check collision
7	-	No collision in the jump direction
8	-	Does jump and remove one charge

Alternative flow: The participator doesn't have charges but is on ground. No collision does jump.

	Actor	**System**
1	Presses the ”Up Arrowkey”	-
2	-	Check so the participator has doublejump charges
3	-	No charges.
4	-	Check so the participator is on the ground
5		On ground
6	-	Check collision
7	-	No collision in the jump direction
8	-	Does jump.

Movement

Summary: The participator presses one of the movementbutton “arrowkey left/right” and checks for collision before moving.
Priority: High
Extends: In-Game
Includes: None
Participators: Character, Enemy
Normal flow of events: The participator tries to move but collides with a worldObject.

	Actor	**System**
1	Click a movement button (Right/Left - Arrowkey)	-

2	-	Check for collision
3	-’	Stop Movement in x-direction

Alternative flow: The participator tries to moves.

	Actor	**System**
1	Click a movement button (Right/Left - Arrowkey)	-
2	-	Check for collision
3	-’	No collision the participator moves.

Collect coin

Summary: The character is within range to collect coin
Priority: High
Extends: In-Game
Includes: None
Participators: Character

	Actor	**System**
1	Actor close to coin	-
2	-	Removes coin from world
3	-	Adds the coins to your current number of coins.

Take Damage

Summary: The player takes damage.
Priority: High
Extends: In-Game
Includes: Die
Participator: Character, Enemy
Normal flow of events: Participator takes damage and survives

	Actor	**System**
1	Actor takes damage	-
2	-	Calculates health left after damage

Alternative flow: Participator takes damage and despawns.

	Actor	System
1	Actor takes damage	-
2	-	Calculates health left after damage. The health is below 0.
3	-	Check if the participator is character.
4	-	The participator isn't character. Despawn the participator.

Alternative flow: Participator takes damage and dies.

	Actor	System
1	Actor takes damage	-
2	-	Calculates health left after damage. The health is below 0.
3	-	Check if the participator is character.
4	-	The participator is the character. The character dies.

Die

Summary: The character dies.
Priority: High
Extends: In-Game
Includes: None
Participator: Character

	Actor	System
1	Actor dies.	-
2	-	Transport to mainmenu after saving.

Shop

Summary: The user presses tab to open the shop
Priority: High
Extends: In-Game
Includes: None
Participator: Character
Normal flow of events: Opens up the shop

	Actor	**System**
1	Presses tab	-
2	-	Pauses the game and opens the shopview

Alternative flow: If the shop is already open close it and unpause.

	Actor	**System**
1	Presses tab or clicks on return button	-
2	-	Unpauses the game and close the shopview

Buy upgrade

Summary: Tries to buy an upgrade level if the cap isn't reached and the number of coins is sufficient.
Priority: High
Extends: Shop
Includes: None
Participator: Character
Normal flow of events: Tries to buy upgrade but cap is already reached.

	Actor	**System**
1	Left mouse click on the upgrade	-
2	-	Check if the adding one level would exceed the cap.
3	-	It would, don't add level

Alternative flow: Tries to buy upgrade but not enough coins.

	Actor	**System**

1	Left mouse click on the upgrade	-
2	-	Check if the adding one level would exceed the cap.
3	-	It wouldn’t.
4	-	Check if enough coins.
5	-	Number of coins isn’t sufficient.

Alternative flow: Buys upgrade a level

	Actor	**System**
1	Left mouse click on the upgrade	-
2	-	Check if the adding one level would exceed the cap.
3	-	It wouldn’t.
4	-	Check if enough coins.
5	-	Number of coins is sufficient.
6	-	Add one level to the upgrade