ASCENT GAME DESIGN DOCUMENT

Prototype 2 – Celeste Clone

A platformer about climbing out of the underworld as you learn to persevere, learn and overcome.

Prototype 2 – Team 6

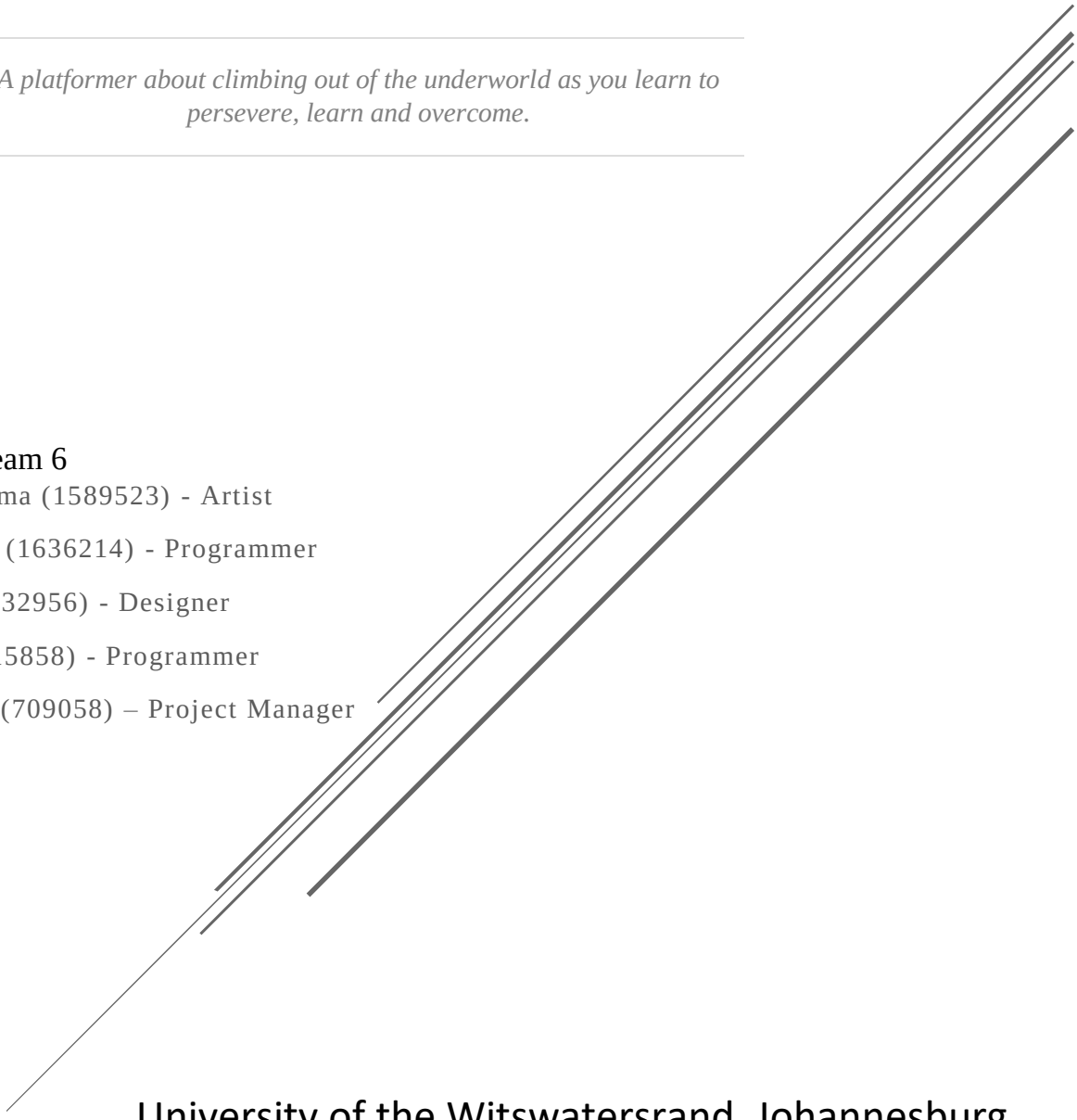
Dominique Bothma (1589523) - Artist

Keaton De Jager (1636214) - Programmer

Stash Gomes (1532956) - Designer

Devan Gray (1615858) - Programmer

Matthew Morris (709058) – Project Manager



University of the Witswatersrand, Johannesburg

Game Design IIIB (WSOA3003B)

Contents

1. Game Overview

1.1. Game Concept

The prompt for this project was to create a clone of the hit Indie game Celeste (2018). Our rendition will focus on duplicating the core mechanics of Celeste and providing a new fresh art style. The purpose of this prototype serves as a technical and design exploration of the original game; and an exploration of a new art direction.

1.2. Key Features

The game has the most basic components of movement present in the Celeste game. This includes side to side movement, jumping, wall grabbing, wall climbing, wall jumping and dashing. Additionally, the game has a basic implementation of levels, particle effects and sound.

2. Mechanics, Dynamics and Aesthetics

2.1. Mechanics

2.1.1. Controls

Function	**Keyboard**	**Controller**
Moving (also directs the dash)	WSAD	D-Pad or Left Analogue Stick
Jump	C	A or Y
Dash	X	X or B
Grab	Z or V or Left Shift	LB, RB, LT or RT

Table 1 - Ascend's Controls

2.1.2. Basic Movement

Basic Left and right movement - This is precise. It sets velocity. There is very little ramp up time, with an even shorter deceleration. Needs to be a check for if there is a wall currently in front of the player (while grounded) to prevent them from walking into the wall. If the player is airborne and walking into the wall, they enter the sliding state.

2.1.3. Jumping

Jumping – Should include the "hold to jump higher". If the player presses the jump key but can't currently jump (due to being airborne) then the jump is queued for a very short duration. If the player then meets the criteria to jump, such as sliding down a wall or being grounded) then they jump. This way the player can press the jump key just before landing but still be able to jump. There also needs to be some coyote time, where the player can jump shortly after walking off a ledge.

The player jumps about 1,5x their height, but can jump up to 2.5x their height when held down.

2.1.4. Wall Sliding

If the player is airborne **but** moving towards a wall, they slide down the wall, falling at a reduced speed. Letting go of the movement will return the player to their falling state. Holding the grab key will place the player in the grab state.

2.1.5. Wall Grabbing

If the player is next to a wall, either falling, grounded, dashing or sliding, they will grab onto the ledge stopping all movement.

2.1.6. Wall Climbing

While a player is grabbing onto a ledge, using the up or down keys they may move along the edge.

2.1.7. Wall Jumps

1. If the player is falling next to a wall, the jump will push them up and away from the wall a short distance. Holding the jump key longer still increases jump height, but not the distance the player is pushed from the wall.
2. If the player is sliding down the wall, jumping will push them up and away from the wall. The push away will be a greater distance. Holding the jump key longer still increases jump height, but not the distance the player is pushed from the wall.
3. If the player has grabbed onto the wall, then jumping will push them up the wall a small distance. Holding the jump key increases the jump height. This works by temporarily disabling the grab ability.
4. If the player has grabbed onto the wall and facing away from the wall (diagonally included) then they jump as if it's case 2.

2.1.8. Dashing

The player can dash only once until they are grounded again.

1. While airborne - the player can press dash to shoot in one of 8 directions (up, down, left, right, and diagonals). This is determined by which direction the left joystick is currently being pressed. If there is no directional input, then the dash defaults to the facing direction. If the player is holding grab while dashing and then collides with a wall they immediately grab. Onto that wall.
2. If the player is grabbing onto the wall - the player still dashes as normal. However, grabbing is disabled and reenabled at the end of the dash. This allows players to dash off walls while grabbing or dashing further up a wall.

2.1.9. Death and Checkpoints

When a player dies, either by colliding with spikes or falling out of the level bounds, they will immediately respawn at their last checkpoint. Checkpoints are determined by which area the player enters from.

2.1.10. Jump Pads

When a player lands on a jump pad they are launched up in the air, maintaining their x velocity. Additionally, the pad acts as ground, refreshing the player's ability to dash.

2.1.11. Spikes

Static traps which the player needs to overcome. When the player falls onto a set of spikes the player is killed and respawns at the start of the level. Spikes can appear on any surface within the game, such as the floor, walls and ceiling of levels.

2.1.12. Collapsing Platforms

Dynamic platforms which break after a little while after the player has collided with them. These platforms do reappear after a short time or in the event the player dies.

2.2. Dynamics

2.2.1. Learning through failure

The game is designed to be difficult. The implementation of precise mechanics allows for the development of challenging levels. This means that the player could possibly fail many times before passing a level. Levels are designed such that, even though they are tough, the player understands where they failed and how they can improve to complete the level.

2.2.2. Experimentation as Tutorialisation

Ascent has a vast collection of mechanics. The mechanics are gradually introduced through level design. There is at maximum only 1 new mechanic introduced in a single level. Thus, the complexity is increased gradually, and the player should not feel overwhelmed.

Another part of Ascent's level design is that levels are created with multiple paths/ solutions to complete the levels. This allows players to experiment and try different strategies to finish a level.

3. Level Design

The 3 levels present in this prototype were adapted from 3 existing levels in Celeste. This was done due to time constraints, but also provided us as developers with the opportunity to tune our player controller to be more in line with the original, as well as to compare how our controller feels in comparison to the original in a like to like comparison.

3.1. Level 1

The First level is meant to introduce the player to the jump, the held jump and the dash. This is because of how the jumps are spaced.

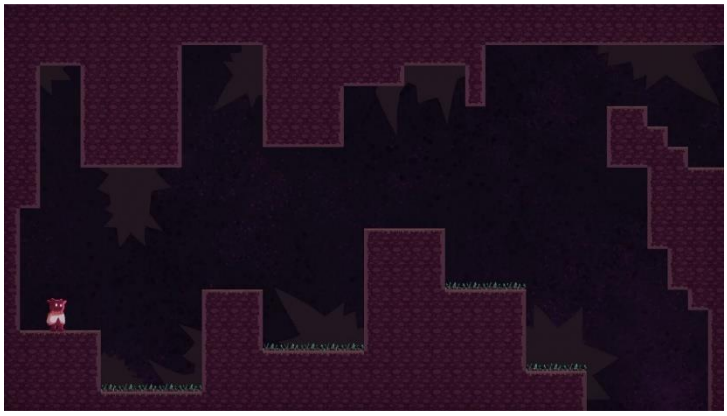


Figure 1 - Ascend Level 1

The first jump should be enough for the player to just hop over.

The second jump requires the player to hold. If they fail to hold the jump key, they will enter a slide and discover wall jumping.

The third jump requires the player to dash and grab.

Additionally, the player may accidentally discover they can climb on the wall. If so, they may also discovery or experiment with wall jumping on the left most wall. If they manage to climb further up, they will discover they can repeatedly jump between walls.

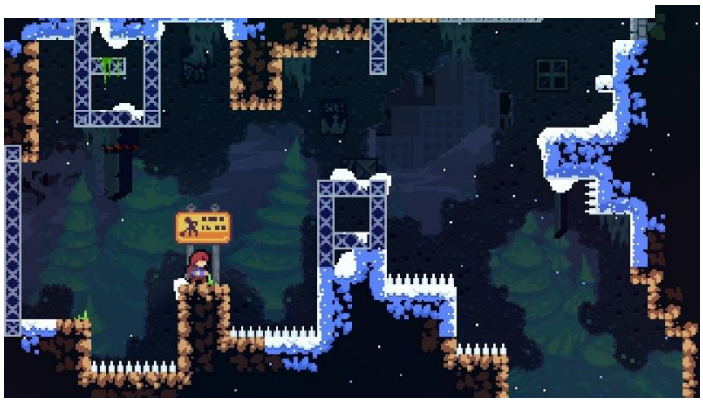


Figure 2 - Celeste Level 1

From the onset, this level should establish that you will die frequently in this game, but not be punished too severely for it

3.2. Level 2

This level introduces the player to the jump pad. As well as the concept of using the dash to quickly change the player's direction to perform more precise movements.

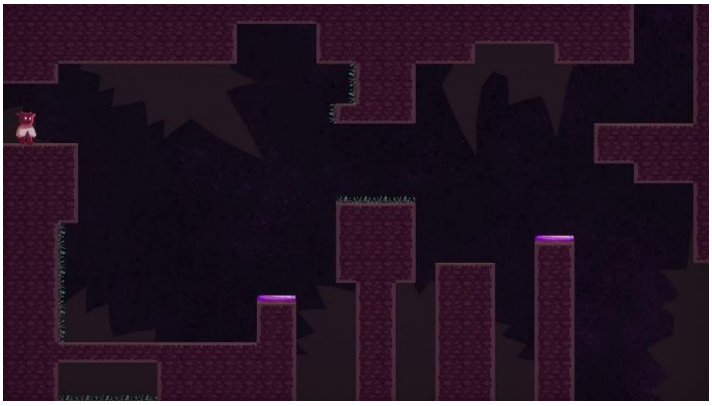


Figure 3 - Ascend Level 2

The level intentionally only has one difficult section. This is to focus the player on this specific task without demanding much more from the player. This limits frustration, as the player only needs to perform one task, then complete the level.

In the previous level, the player is forced to perform 3 vital tasks, failing any, sends them right back to the beginning, forcing them to rehearse these tasks repeatedly. However, by this point, the player just needs to understand how jump pads work, and that dashes can be used to redirect the player, or interrupt movement. This is important for the next level.

Figure 4 - Celeste level 2

3.3. Level 3

The purpose of this level was to introduce the player to breakable platforms and using the climbing ability to provide more time for the player to plot their next move.

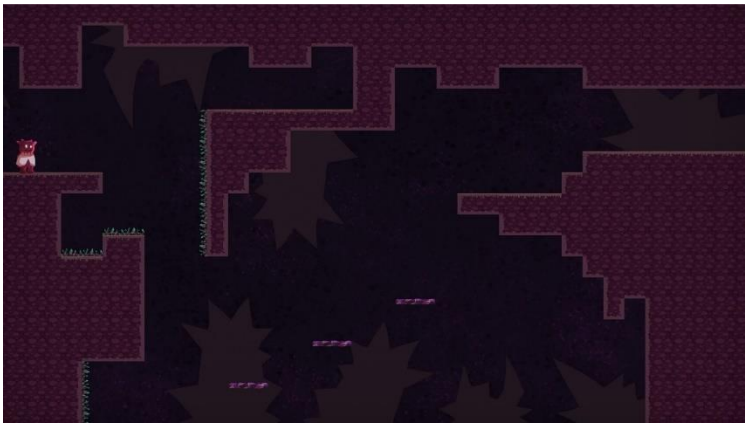


Figure 5 - Ascend Level 3

Additionally, the player is required to dash to reach the first breakable platform. From here, the player learns that these thin platforms break.

The player may also learn that they can grab onto the left wall and jump off it instead of dashing.

Multiple solutions to these levels are important, as they give room for the player to come up with their own solutions to the puzzles they face, rather than having to search for the one intended solution.

Figure 6 - Celeste Level 3

4. Visual art, narrative and Sound

4.1.Character Art

- Walk Cycle
- Idle Loop
- Wall Grabbing Idle
- Wall climbing
- Wall Sliding
- Jumping (falling and rising)
- Dashing (you can make up, down, forward, downward-forward and upward-forward variants if you want to be fancy but run this by Keaton first to see if that's possible).

Figure 7 - Ascend's Character

4.2. Level Art

- Tilemap
- Breakable Platform
- Spikes
- Jump Pad

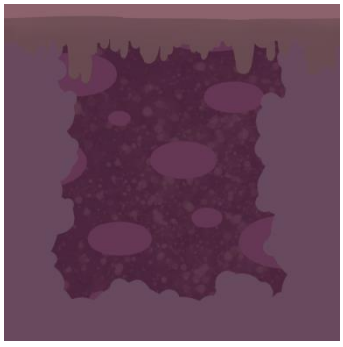


Figure 8 - Ascend's Tilemap

Figure 9 - Ascend's Breakable Platform

4.3. Particle Effects

- Dash Effects
- Landing effect
- Jumping effect

Figure 10 - Ascend's Jump Pad

4.4. Sounds

- Jump sound
- Land sound
- Dash sound
- Death sound
- Revive sound

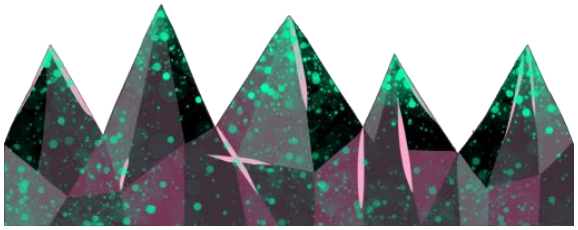


Figure 11 - Ascend's Spikes

5. Technical

5.1. Hardware

All members of the group will be using Dell Inspiron 15 7577 laptops for development with the following specifications:

- Intel 7th Generation Kabylake Core i7-7700HQ processors.
- 16GB of DDR4 RAM.
- 256GB ultra-fast SSD.
- 1TB hard drive.
- NVIDIA GeForce GTX 1060 6GB GDDR5 graphics cards.

5.2. Software

5.1.1. Unity (Version 2019.1.10f1 (64-bit)) as the Game Engine for development

Unity has been chosen as the game engine used for the development of this project due to all group members being experienced in using the software.

5.1.2. GitHub

GitHub allows for easy integration and sharing between all members of the team. It also allows the team to assess what has been implemented and what hasn't at each stage of the project. GitHub also allows us to roll back to an earlier stage of the project, should anything go wrong.

5.1.3. SourceTree

The GUI application that the group will use to work with git to manage and handle the main repository used to produce this prototype.

5.2.4. Visual Studio Code

Visual studio code provides syntax highlighting, intelligent code completion both which will benefit our programmers in being more efficient. Visual studio code also assists in debugging and has embedded git integration.

5.3. Class Diagrams

5.3.1. Environment Scripts

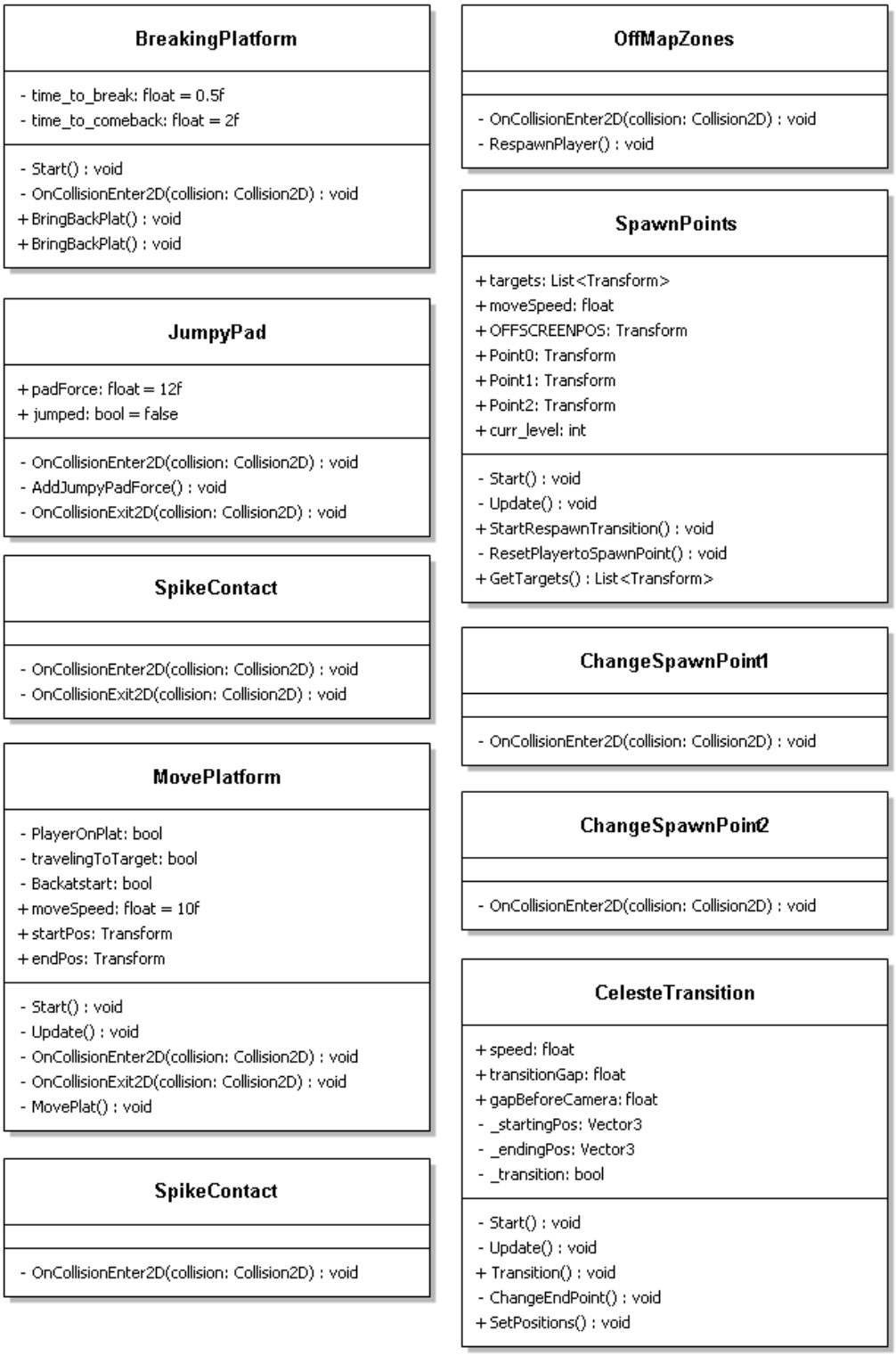


Figure 12 - Environment Scripts Class Diagrams

5.3.2. Player Movement Scripts

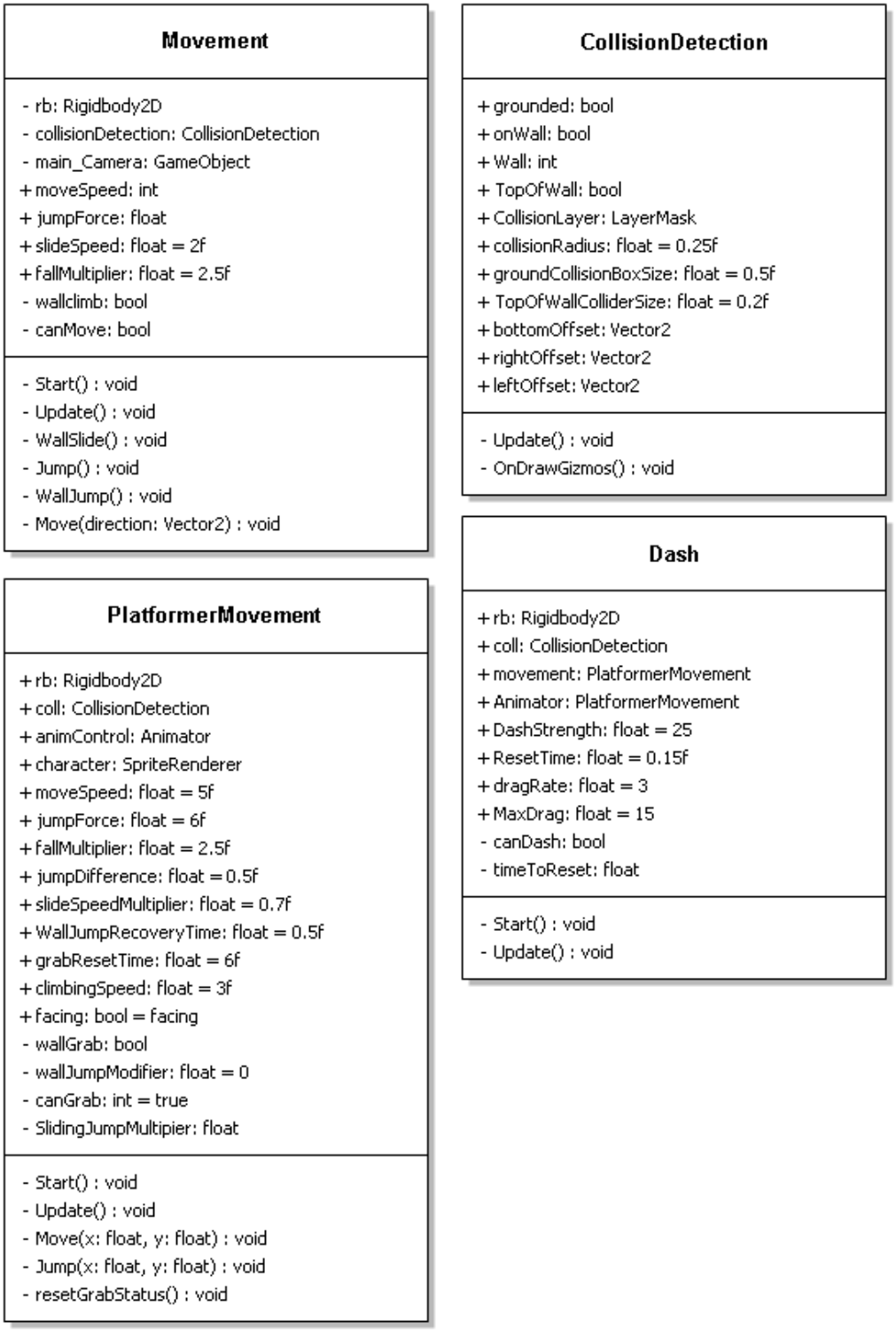


Figure 13 - Player Movement Scripts Class Diagrams