FoGD Project report – Jeremias Wahlsten

Introduction

This is the project report for the final project for the course Fundamentals of Game development. Inside this report I will talk about the technical details regarding the game, assets used and their origin, grading the course and finally my own thoughts and conclusions. I also have a chapter that disclosed the use of ai in this project at the bottom.

The game is a 2d platformer partly inspired by games like super meat boy, where you have semi-precise control of your character and must beat difficult levels one after another. However, due to this being a final project for a course, the scale is obviously much smaller and some of the assets are imported.

Technical details

In this chapter, I will be talking about the technical details regarding all software used, and the I will explain the core systems inside the game.

Base details

The game was made with Godot v4.4.1 Stable edition, without C# support. All code is made using Godot own programming language GDScript. For recording and modifying audio files I used Audacity. Git and Github was used for version control and upload to Codegrade. Paint.net was used to modify assets, like making a custom spritesheet.

Game's core system

Player movement System

I made the movement based on the Platformer toolkit, where the player's movement is governed by several variables that allow for fine-tuning:

- speed (600.0): Maximum horizontal movement velocity
- jump_speed (-1000): Vertical force applied during jumps
- acceleration (5000): Rate of speed increase when moving
- deceleration (7000): Rate of speed decrease when stopping

- mass (2.3): Abstract value that affects gravity's impact on the player

The jump system adapts dynamically to gravity direction, allowing for both normal and inverted movement when powerups are active. Visual feedback is handled through state-based animations that respond to movement direction, airborne status, and damage states, with sprite flipping providing clear directional indicators.

Enemy AI Behavior System

The enemy AI operates through a state machine with four behaviors:

- Idle: Enemies monitor their surroundings for player detection within a defined range. Upon detection, they transition to chase state. This detection works even through walls, cause bats use echolocation
- Chase: In chase, enemies linearly move towards the player's position while maintaining pursuit boundaries. When players escape beyond the view range, enemies enter a return-to-spawn state.
- returning to spawn: enemies start navigating back to their original positions using linear vectors. If the player returns then enemies start again, but if they reach the spawn, they start the Idle state.
- Dying: The dying state handles defeat scenarios with proper collision cleanup and visual effects.

Powerup and Gravity Systems

The powerup system implements a three-phase activation model: collection, activation, and cooldown. First players must find and reach a powerup, which enables the powerup and removes the collectible from the map. Activation triggers a temporary gravity reversal. Powerup activation has a cooldown period to prevent exploitation in level puzzles.

During gravity reversal, the entire physics system adapts - jump mechanics invert, collision detection adjusts for ceiling-walking, and visual elements flip to match the altered physics state. The enemies can also deactivate the powerup if the player gets hit while the powerup is active.

HUD and Death screen systems

The HUD adds hearts based on the max health set up and updates them every time player’s health changes. This system allows a modular change in the player’s health if needed. The health changes also have a simple animation on it.

The Death screen is included in the HUD, but originally hidden. If the player dies, then the death screens becomes visible with a little effect. It pauses everything inside the root tree, except the HUD.

Level change System

The level changing is handled entirely by the game scene and its script. It originally has the first level loaded, but any level number can be loaded from the levels folder. As long as all levels follow the correct naming, all of them can be loaded. To easily keep track of where the current level is (cause new child scene goes to a new place, which is annoying to deal with in code), the game scene has a child called “Level_current”, which has the level as its only child. This way the current level can always be easily found

Assets Used

Textures

for the textures I used Kenney’s Platformer Art Deluxe Pack, which is a great 2D art pack that uses the CC0 license. This pack provided me with free to use, great quality assets with a consistent and coherent visual look needed for points.

Sounds

For sounds effects I used audacity to record all sound effects.

The background Music was made by my friend for our game in Finnish game jam 2025. He has given me to re-use his music for this game as well.

Other

additionally, some experimentation, studying and testing like the characters speed was adjusted by using an interactive video essay called “platformer Toolkit”, which is from the creator of **Game Maker’s Toolkit**. This essay can be found in itch.io from the link.

Points

the features and points have been taken directly from the project work description

The game can be played, it does not crash, the player does not get stuck, etc.	2
The game has consistent and coherent visual look	2
The music and sound effects are in balance	2
The game has setting screen, where gamer can customize settings (e.g. music and sfx volume)	2
The game has enemies that can be destroyed	2
The enemies have “some intelligence” (state machine is enough)	3
There are some collectable items in the game (e.g. coins, ammo, guns, starts…)	1
The game area is bigger than just one screen	1
There are different menu and game scenes	1
There are various maps with increasing difficulty	2
Gamer can see how fast/good she passes the level/map	1
All the settings and game records (high score / TOP10, passed levels, etc) are stored in a save file(s)	2
Finishing something (eg. a map) unlocks something else (e.g. another map)	2
Physics engine is used in innovative way (e.g. there are areas where the gravity changes or some enemies have reversed gravity, etc)	2
The game supports various control options (e.g. keyboard and mouse / touch screen / gamepad)	2
The game has dynamic lightning (i.e. the lightning varies on different areas / times of the game)	3
Summa	**30**

Conclusion

All in all, I’m personally happy with how this project turned out. I basically made this whole project in 2-3 days, which felt like a solo game jam that I liked. I had many ideas to expand this game forward like a level select screen, level-specific high scores, scoreboard for all high scores, more enemies and powerups and much more.

But its not all positive. Due to making this in such a short time, I’ve discovered multiple bugs that should be fixed, like if you activate your power early in the level (like in level 3), it will get cancelled prematurely. There are also some design issues, like how you are supposed to jump on a bat’s head if the gravity is reversed.

But for now, this project is done, and it should give me full points regarding the course. I hope that this project reflects my ability to make games well enough to guarantee a grade 5, which I’m aiming for. I also left my personal best inside the executable game file, if you dare to beat it. Good luck!

Declaration of AI

The following generative AI tools were used in this project:

- ChatGPT (free license):
  - brainstorming
  - document text refinement
- Github copilot (student licence):
  - generating commit messages
  - debugging
  - auto-complete (mostly for comments or boilerplate)