# TURBO RUNNER’S GAME

# Abstract

Turbo Runners is a 2D platform game developed in Python using the Pygame library. The project was designed to deliver a fully functional game with three levels of increasing difficulty, interactive controls, and a clear win or lose outcome. Players control Blippo, who must move across platforms, avoid hazards, collect coins, and use power-ups to defeat Lord Zing and rescue a captured friend.

Each level is subject to a timer and begins with three lives. The player can run faster by holding the mouse button, jump by clicking or pressing the Space key, and shoot projectiles using the Enter key. Menus include a main title screen with instructions, a level selection screen offering Easy, Medium, and Hard modes, and end screens showing analytics after each game. Analytics include score, coins collected, deaths, time played, and, in the case of victory, a performance rating.

The project followed a structured process of requirements analysis, investigation, design, development, testing, and evaluation. The final product is a complete and playable game that meets the specified requirements, provides a consistent user interface using the Forest Green and Deep Blue colour scheme, and demonstrates the effective application of Python and Pygame for interactive development

Contents

[TURBO RUNNER’S GAME i](#_Toc208282531)

[Abstract i](#_Toc208282532)

[Introduction 1](#_Toc208282533)

[Identify 4](#_Toc208282534)

[Investigation 6](#_Toc208282535)

[Requirements 9](#_Toc208282536)

[**System Architecture** 11](#_Toc208282537)

[**Player Design** 13](#_Toc208282538)

[Platforms and Obstacles 17](#_Toc208282539)

[**Collectibles and Power-Ups** 20](#_Toc208282540)

[**Boss and Friend** 22](#_Toc208282541)

[**Heads-Up Display (HUD) and Analytics** 24](#_Toc208282542)

[Development 26](#_Toc208282543)

[**Stage 1: Setup and Constants** 26](#_Toc208282544)

[Stage 2: Animated Background 28](#_Toc208282545)

[Stage 3: Player 30](#_Toc208282546)

[**Stage 4: Moving Platform and Platform** 35](#_Toc208282547)

[Stage 5: Moving Obstacle (enemies & spikes) 36](#_Toc208282548)

[**Stage 6: Lord Zing** 38](#_Toc208282549)

[Stage 7: Friend rescue system 39](#_Toc208282550)

[Stage 8: Power Ups and Collectibles 40](#_Toc208282551)

[Stage 9: Game (engine, levels, camera, UI) 42](#_Toc208282552)

[9a Initialization & reset 42](#_Toc208282553)

[9b Level creation 43](#_Toc208282554)

[9c Camera & shake 46](#_Toc208282555)

[9d Event handling 47](#_Toc208282556)

[9e Update loop 49](#_Toc208282557)

[9f Menu screen 53](#_Toc208282558)

[9g Level select 54](#_Toc208282559)

[9h Gameplay drawing 56](#_Toc208282560)

[9i HUD 61](#_Toc208282561)

[9j Game Over 63](#_Toc208282562)

[9k Victory 64](#_Toc208282563)

[9l Buttons helper 66](#_Toc208282564)

[Stage 10: Main entry point 67](#_Toc208282565)

[Summary of Development 69](#_Toc208282566)

[Testing 70](#_Toc208282567)

[**Internal Testing** 70](#_Toc208282568)

[**External Testing** 74](#_Toc208282569)

[**Summary of Testing** 75](#_Toc208282570)

[Issues Encountered 76](#_Toc208282571)

[**Evaluation** 80](#_Toc208282572)

[**Conclusion** 81](#_Toc208282573)

[Appendix A – Full Game Code 83](#_Toc208282574)

[Appendix B – Additional Screenshots 84](#_Toc208282575)

# Introduction

The purpose of the Turbo Runners project was to demonstrate the design and implementation of a full computer game using Python. Pygame was chosen because it provides accessible methods for handling graphics, input, collisions, and timing, while remaining straightforward enough to keep the codebase simple and readable.

Turbo Runners places the player in control of Blippo, who must complete three levels of increasing difficulty. Level one introduces core mechanics with static and moving platforms, patrol enemies, coins, and a speed power-up. Level two increases difficulty with more moving platforms, patrol enemies, floating spikes, and additional power-ups such as jump and attack. Level three is the hardest, with tight platforms, more hazards, and a final boss battle against Lord Zing. On defeating Lord Zing, Blippo rescues his friend and completes the game.

The report is structured to provide a professional record of the project. It begins with the identification of objectives, followed by an investigation of relevant tools and design principles. It then outlines the requirements, explains the system's design, and provides a detailed description of the development process, referencing the code itself. The report continues with testing and evaluation, concluding with a reflection on outcomes and recommendations for further improvements.

Main menu with TURBO RUNNERS title, subtitle, and on-screen instructions



Level Select screen with Easy, Medium, Hard options and time limits 120s, 90s, 60s



# Identify

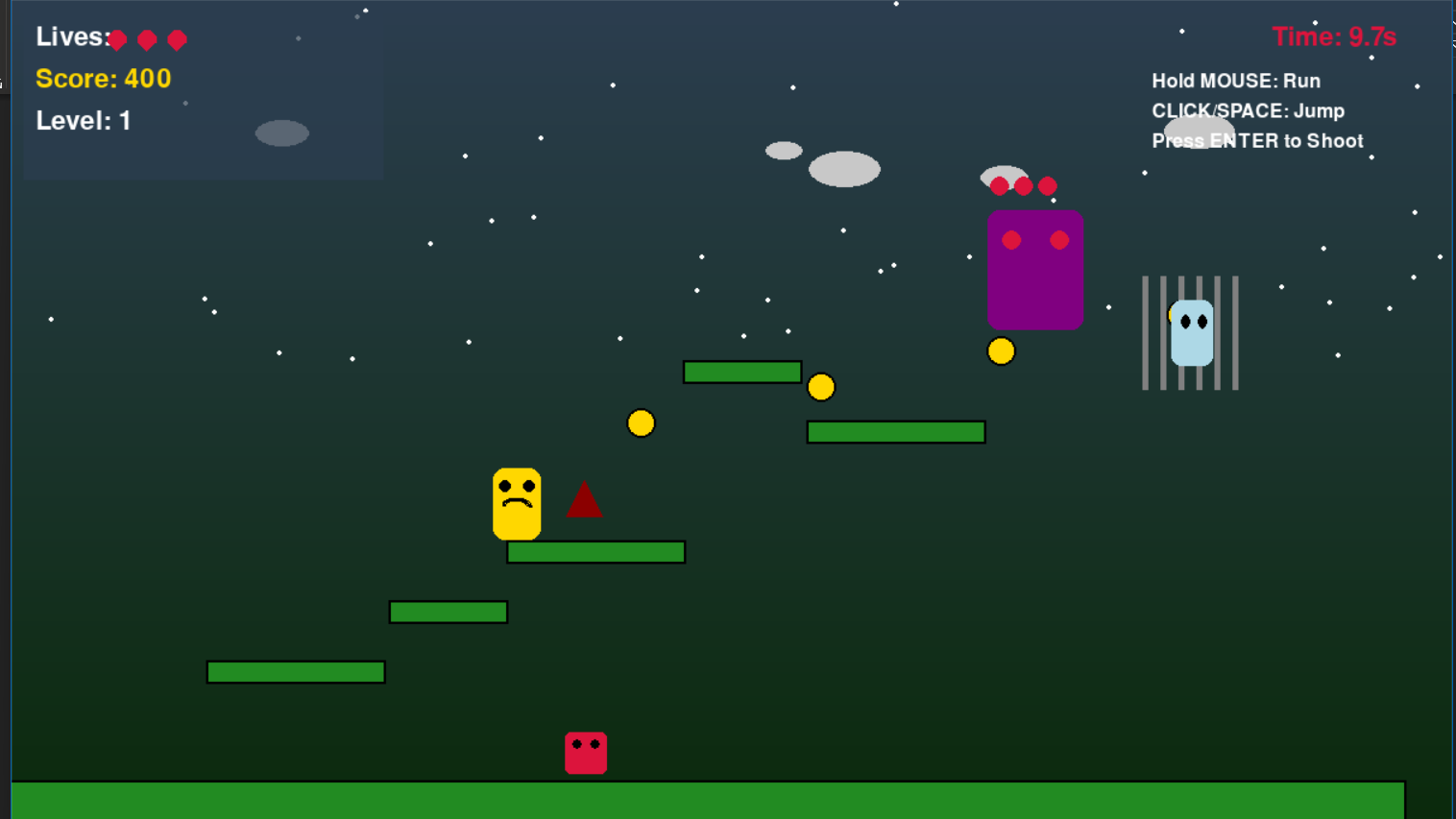
The main aim of Turbo Runners was to build a platform game that was fun to play, technically reliable, and compliant with the given requirements. The challenge was to integrate multiple features — menus, controls, levels, enemies, power-ups, analytics, and a boss encounter — into one cohesive system while keeping the code straightforward and accessible.

The purpose of the game is to allow the player to guide Blippo through three levels to rescue his captured friend. Each level tests the player’s ability to navigate platforms, avoid patrol enemies and floating spikes, collect coins, and use power-ups. The system enforces a time limit and a three-lives rule to ensure that the game remains challenging.

The intended users are casual players running the game on a desktop computer. For them, accessibility and clarity were key considerations. This was achieved through simple menus, clear on-screen instructions, and a heads-up display (HUD) that always shows score, lives, level, and remaining time.

The success of the project was defined by clear criteria: three levels with distinct designs and increasing difficulty, an interactive menu system, a main menu displaying instructions, timers, and lives per level, analytics at the end of each game, and a consistent visual design using the specified colors.

Lives shown as hearts, score, level, timer turning red under 20s, and controls panel on the right



# Investigation

Several areas were investigated before development began.

First, platform game mechanics were analysed. Classic games of this genre rely on smooth, predictable movement, reliable collision detection, and a clear sense of progression. These findings confirmed the need for Blippo to have responsive controls, gravity-based jumping, and accurate platform interactions. The investigation also showed that timers and lives are effective at creating urgency and keeping players engaged, which led to their inclusion.

Second, the choice of tools was confirmed. Python was selected for its readability and suitability for small to medium projects. Pygame was chosen because it provides modules for event handling, collision detection using rectangles, and sprite rendering, all of which were needed for Turbo Runners. Pygame’s ability to update the screen efficiently at sixty frames per second meant that the game would feel responsive without performance issues.

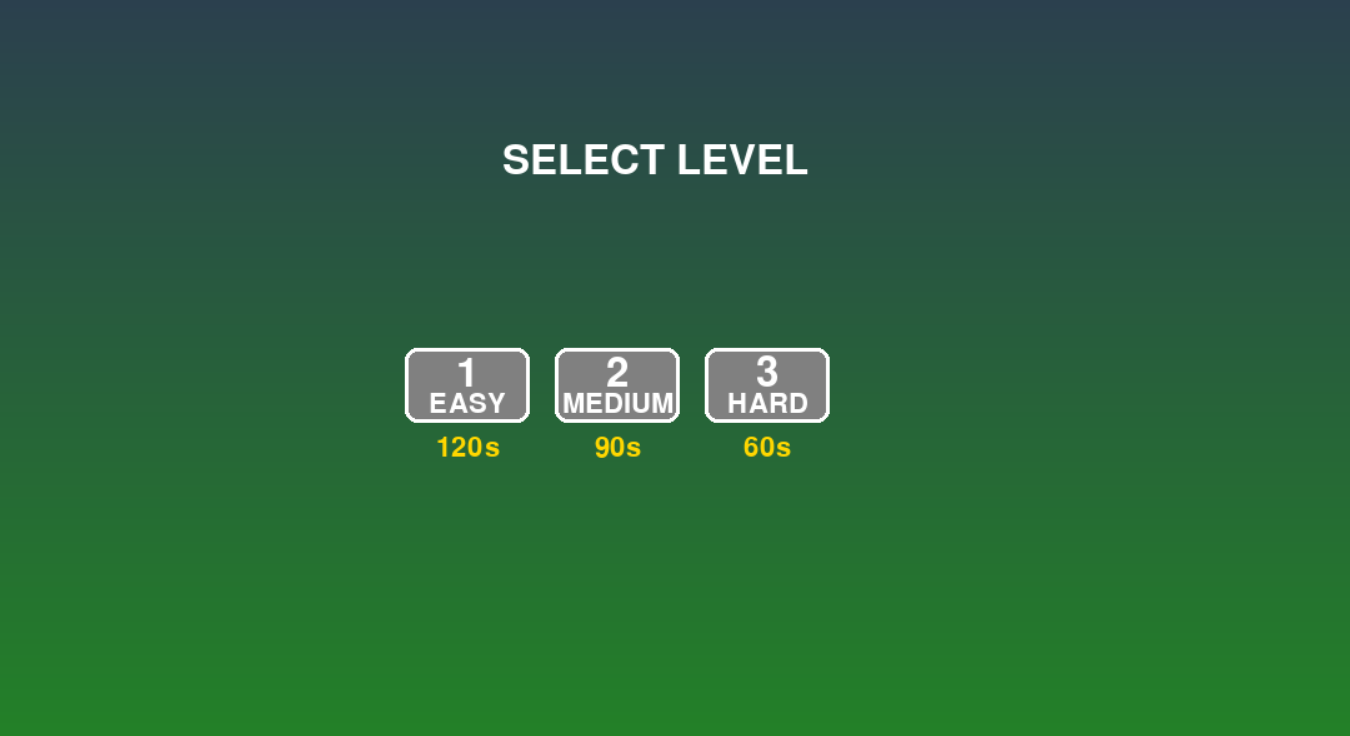
Third, visual design principles were explored. Consistency of colour was highlighted in the project brief, and Forest Green with Deep Blue were tested as primary colours. These colours were applied to menus, HUD elements, and backgrounds using gradient fills and overlays to achieve a professional look. Menu structure was also considered, and it was decided that the main menu would present both the title and the instructions on a single screen, avoiding unnecessary complexity.

Finally, approaches to testing were researched. A combination of algorithm verification, development testing, alpha testing, and beta testing was identified as the best approach. This ensured that individual features such as collision detection could be verified early, and the complete system could later be tested with players for usability and balance.

Background gradient from Deep Blue to Forest Green with static stars and clouds



Menu with instructions visible on the main screen



# Requirements

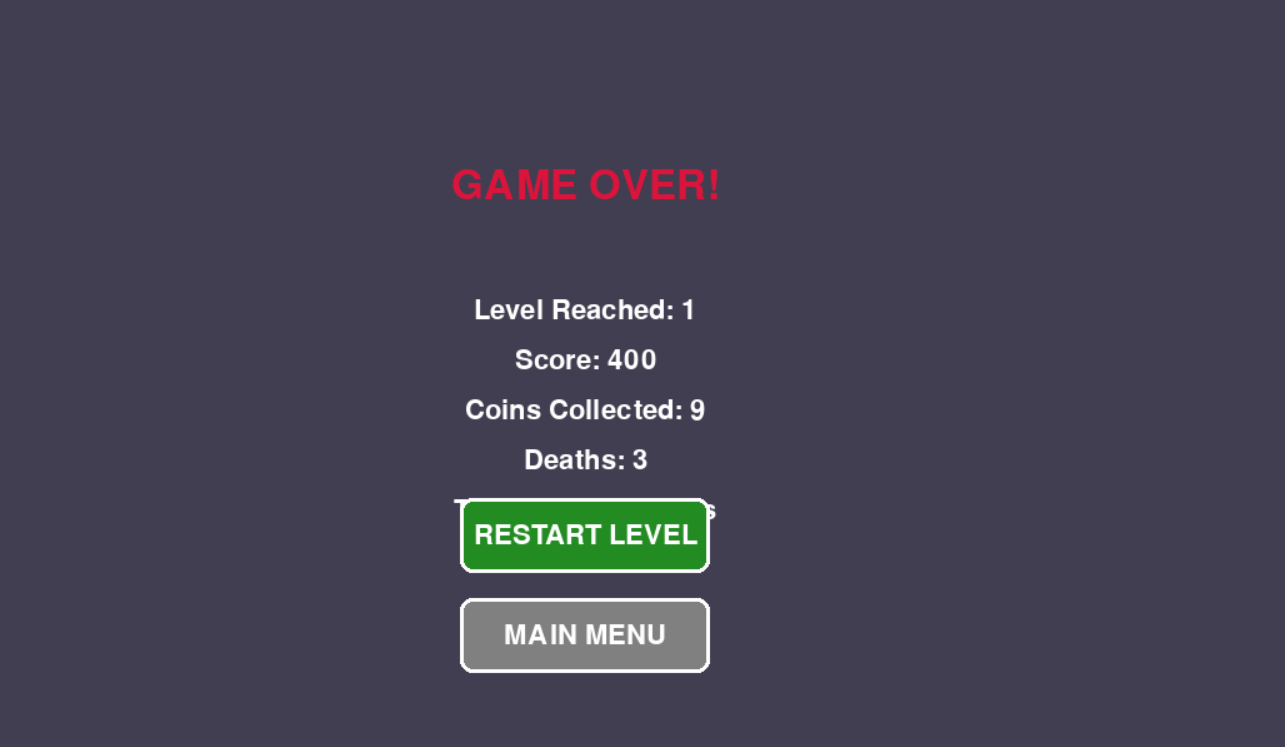
The requirements for Turbo Runners were divided into functional and non-functional categories.

Functionally, the system had to provide three levels with unique layouts, each progressively harder. Level one was to introduce the basics with coins and a speed power-up. Level two was to introduce more hazards and a wider range of power-ups, while level three would culminate in a final boss battle with Lord Zing. The controls had to allow Blippo to run by holding the left mouse button, jump with a click or the Space key, and shoot projectiles with Enter. Each level had to begin with three lives and a timer, and the HUD had to display the remaining lives, current score, level, and countdown time.

The menus had to include a main menu with instructions, a level selection screen with Easy, Medium, and Hard options (corresponding to Levels 1, 2, and 3), and clear Game Over and Victory screens at the end. End-of-game analytics were also required, showing statistics such as score, coins collected, deaths, and time played. In the case of victory, a performance rating had to be displayed.

Non-functional requirements included applying the prescribed Forest Green and Deep Blue color scheme consistently across menus, HUDs, and backgrounds. The code was required to remain simple and commented, avoiding unnecessary complexity. The game also needed to run smoothly at sixty frames per second to ensure responsiveness.

Game Over screen with level reached, score, coins, deaths, and time



The design of Turbo Runners is based on a state-driven architecture, a set of modular entity classes, and a consistent user interface. Together, these elements form the framework that delivers the game experience. This chapter explains the design of the system, describing the architecture, the responsibilities of each class, the layout of levels and interfaces, and how analytics are presented. To aid understanding, relevant code segments are referenced alongside explanations, with placeholders showing where screenshots of code should be inserted.

**System Architecture**

At the highest level, the game is controlled by the Game class. This class manages transitions between states: menu, level select, gameplay, game over, and victory. Each state has its own drawing routine, ensuring that the game reacts appropriately to user input and progress. The Game.run() method contains the main loop, processing events, updating entities, and rendering graphics. This structure keeps the code organised and makes the flow of the game easy to follow.

Game.run() showing the main game loop with state checks



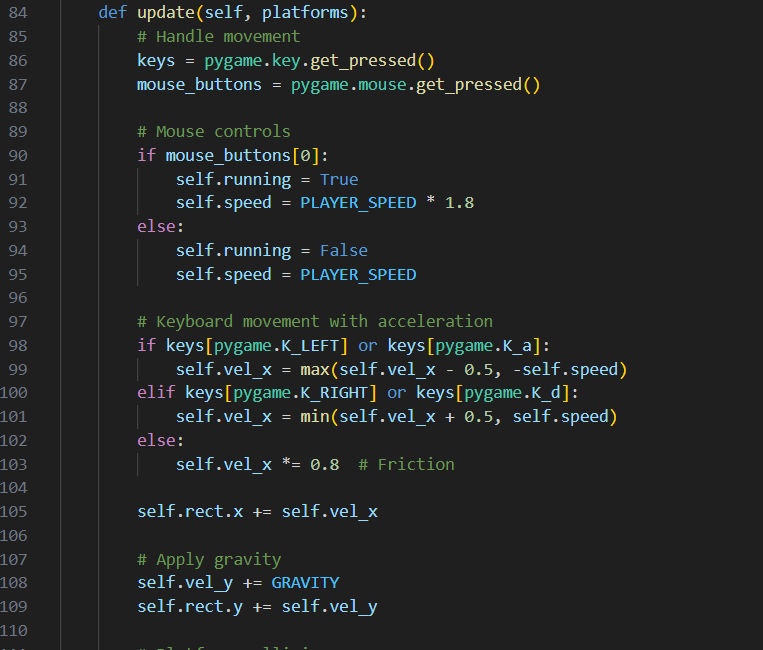
Main menu with TURBO RUNNERS title and instructions

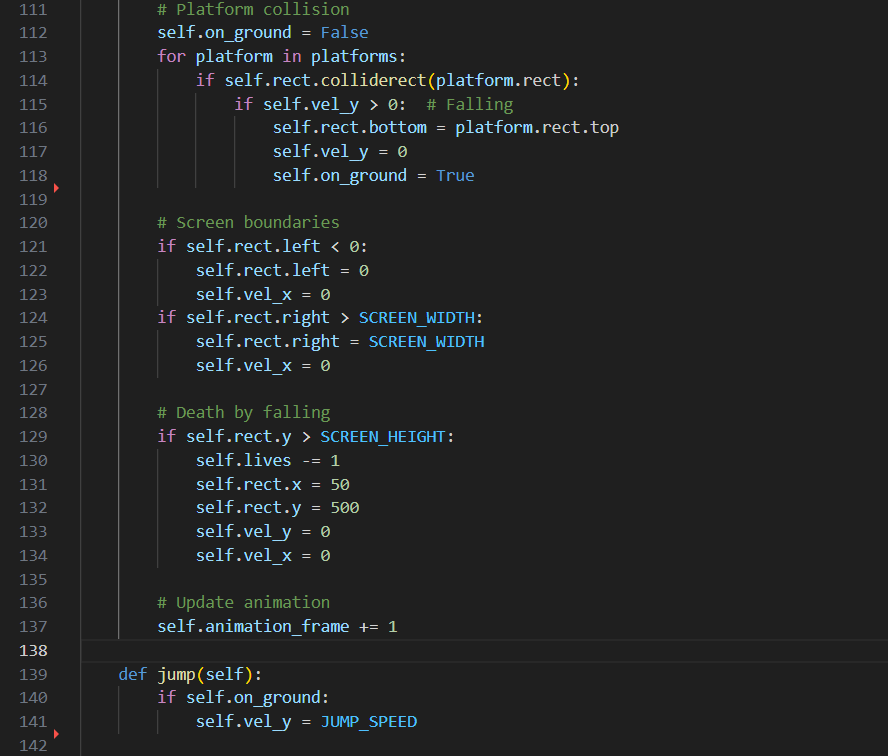


**Player Design**

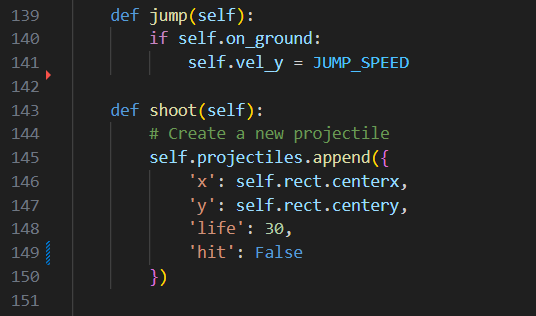
The player is represented by the Player class, which handles input, movement, collisions, and drawing the character. The player can run faster by holding the left mouse button, jump by clicking or pressing the Space key, and shoot projectiles with the Enter key. The Player.update() method applies gravity, updates velocities, handles keyboard and mouse input, and checks for collisions with platforms. The class also includes jump() and shoot() methods for these actions.

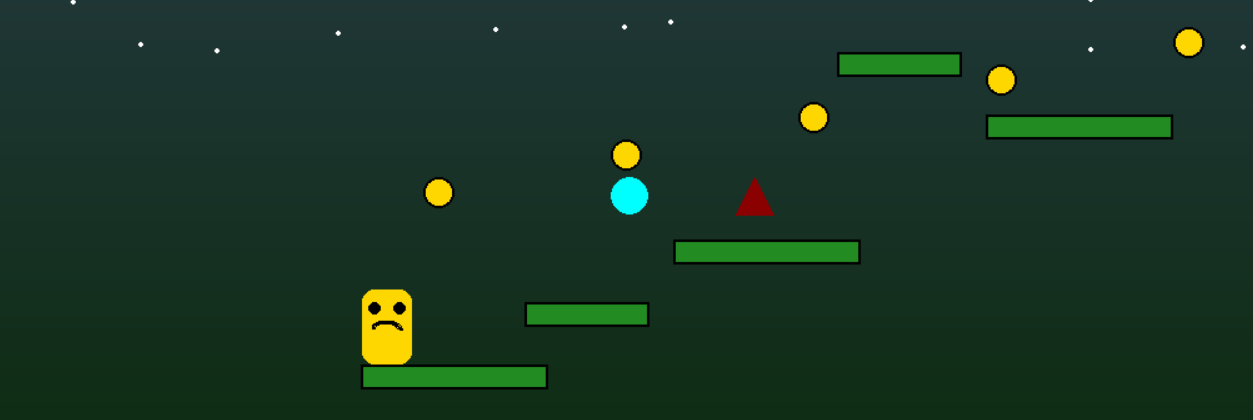
Player.update() showing movement, gravity, and collision checks]





Player.jump() and Player.shoot() methods

  
Level 1 gameplay with Blippo standing on static platforms and coins visible

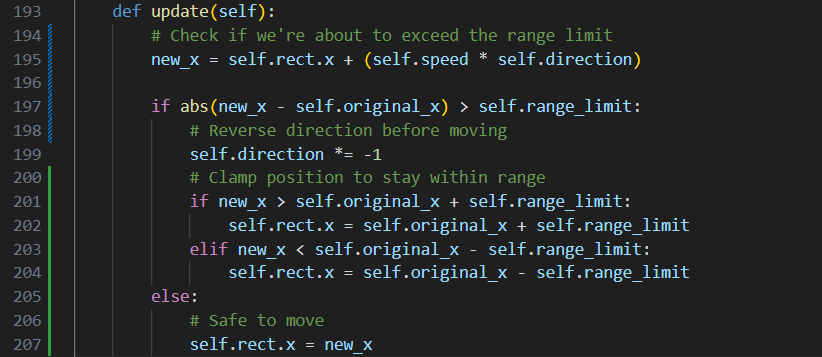


The player’s appearance is drawn as a simple yellow or orange rectangle with facial features. When running, the colour changes and a bouncing animation is applied to give visual feedback. Lives are tracked, and the character respawns if lives remain after falling off-screen or colliding with an enemy.

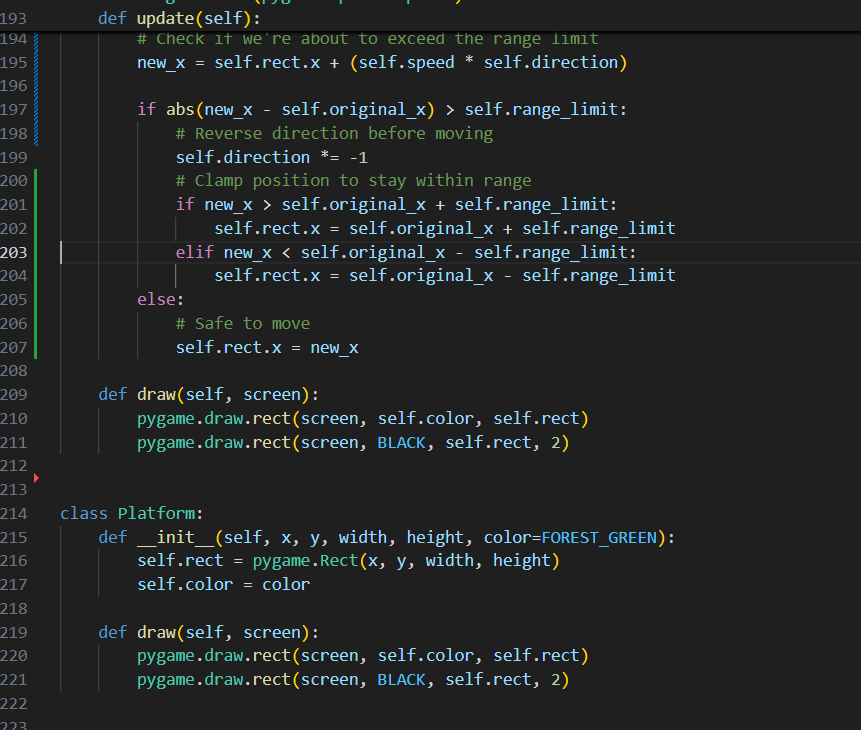
# Platforms and Obstacles

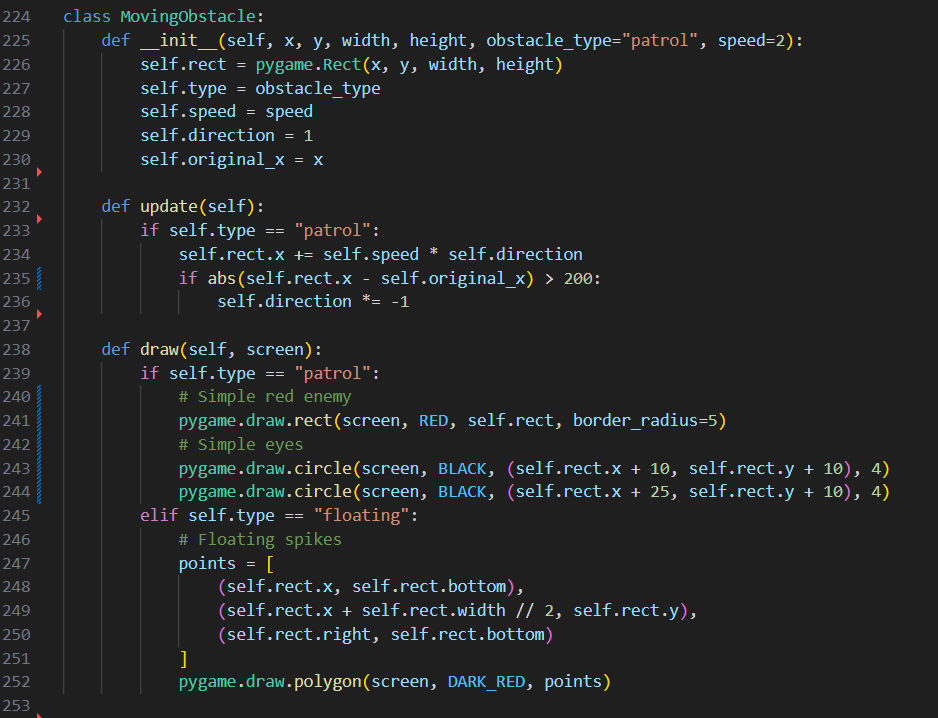
Platforms are represented by two classes: Platform for static objects and MovingPlatform for oscillating surfaces. Moving platforms are restricted to a defined range and reverse direction when reaching it. This adds variety and challenge to gameplay, requiring players to time jumps carefully. Obstacles are represented by the MovingObstacle class, which can either be patrol enemies that move back and forth, or floating spikes that hover menacingly in place.

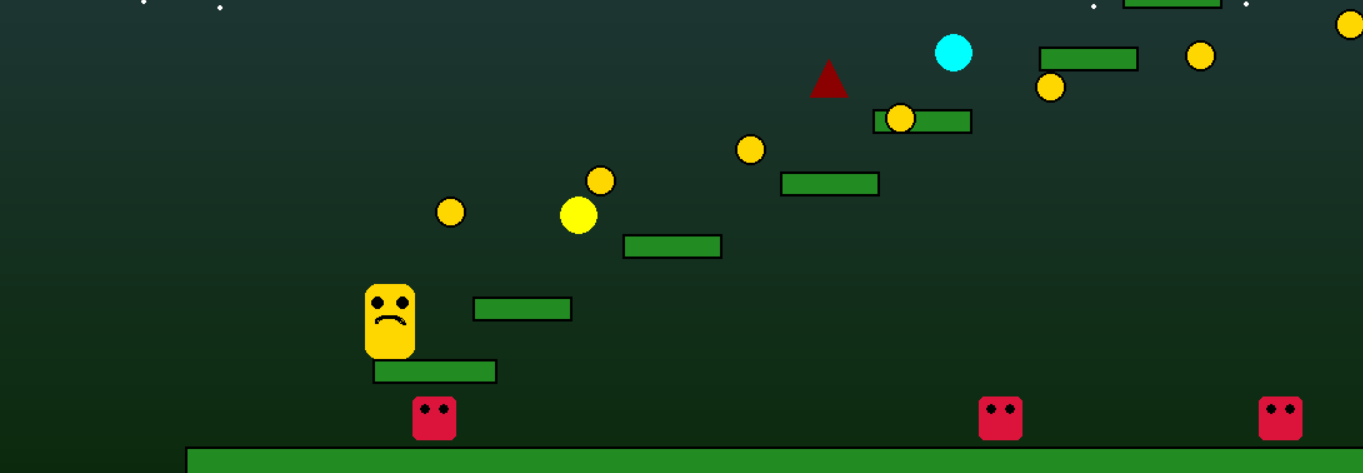
MovingPlatform.update() method showing range reversal



MovingObstacle.update() and draw() methods for patrol and floating types



  
Level 2 gameplay with moving platforms and patrol enemies



These classes are designed to be reusable across levels, ensuring that new challenges can be created simply by placing different instances in different locations.

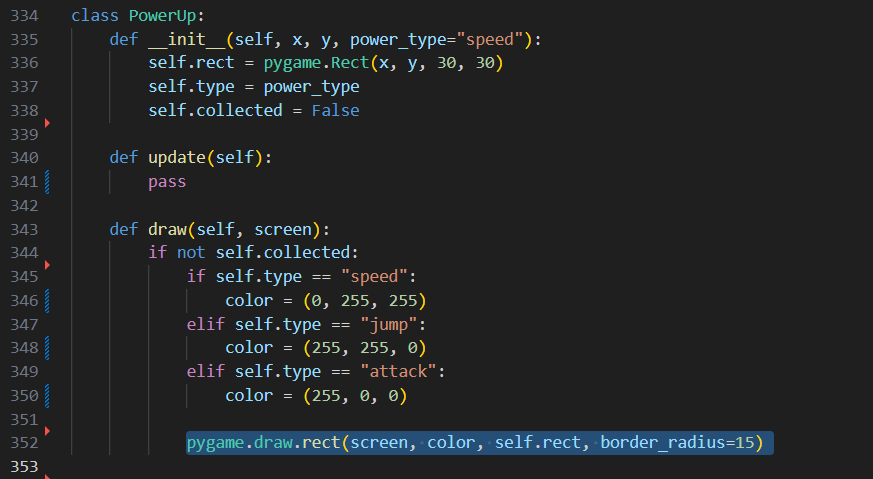
**Collectibles and Power-Ups**

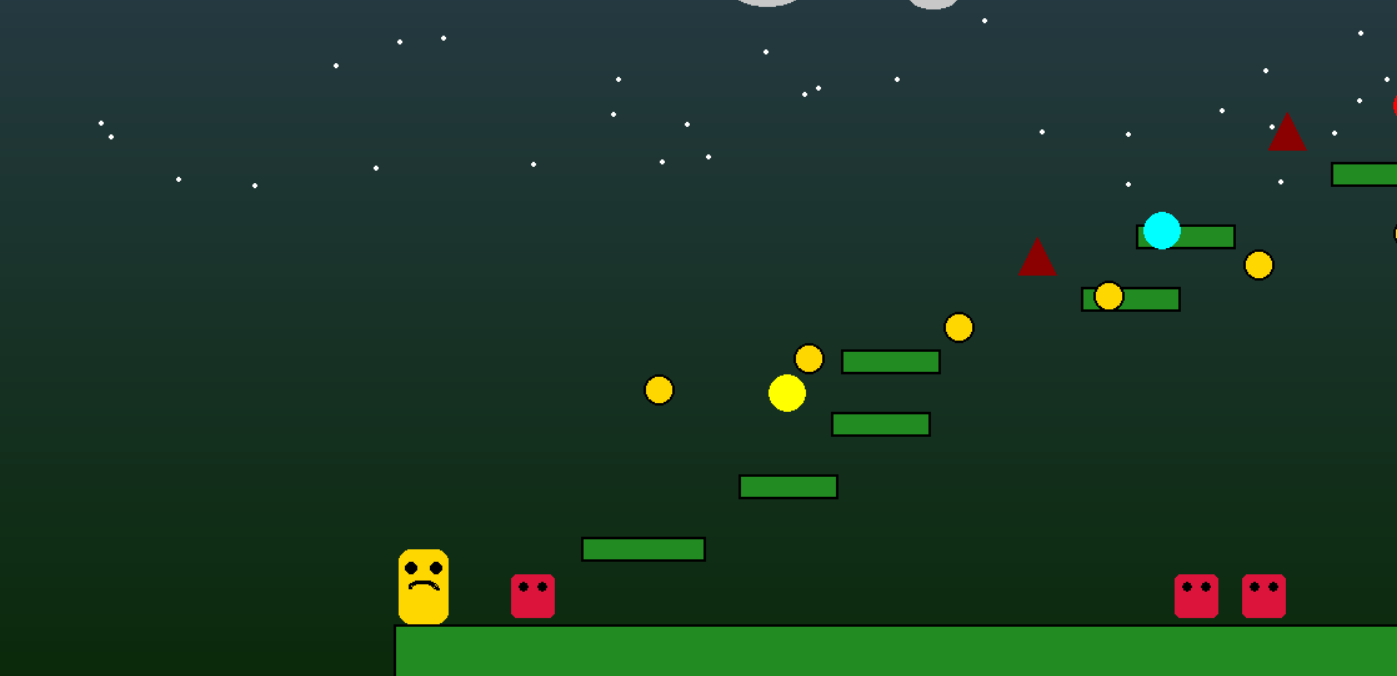
Coins are implemented with the Collectible class. Each coin is a small ellipse that increases the player’s score when collected. The PowerUp class introduces three special items: speed (cyan), jump (yellow), and attack (red). These items reward the player with temporary abilities, such as increased speed or a boost to jump height.

Collectible.draw() showing coin rendering



PowerUp.draw() shows colored types for speed, jump, and attack

  
Level 2 gameplay showing coins and a jump power-up

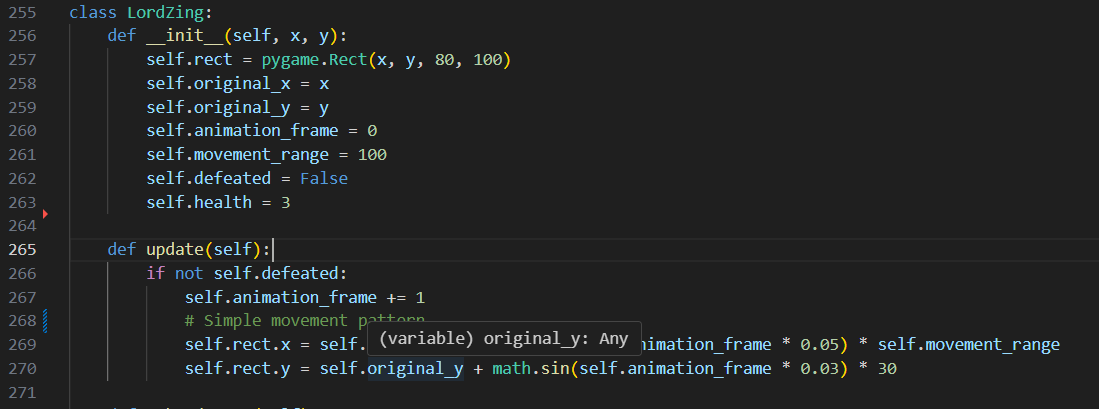


Collectibles and power-ups are scattered across levels, encouraging exploration and rewarding skillful play. Their design also adds variety to the experience, as different runs may yield different outcomes depending on what items are collected.

**Boss and Friend**

The final stage of Turbo Runners introduces Lord Zing, a boss enemy created with the LordZing class. Lord Zing moves in a sinusoidal pattern and has three points of health. Damage can be dealt with projectiles or by close-range mouse clicks. The class includes a take\_damage() method that reduces health and marks him as defeated when it reaches zero.

LordZing.update() showing sinusoidal motion

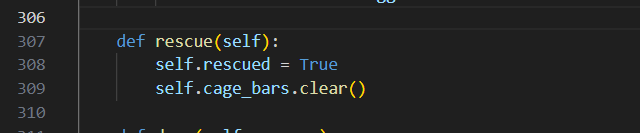


LordZing.take\_damage() showing health decrement and defeat state

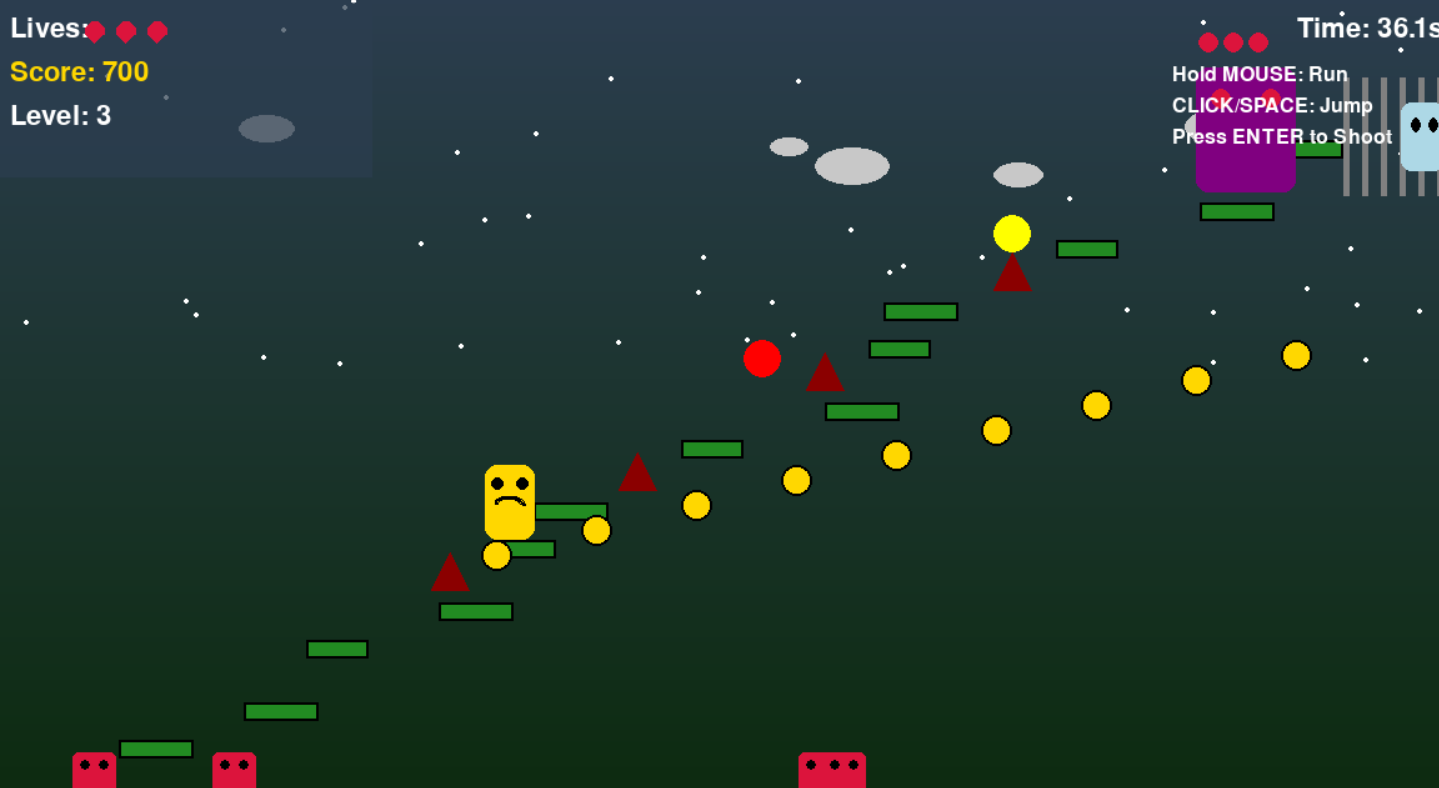


Alongside Lord Zing is the Friend class. The Friend begins trapped inside a cage represented by a set of bars. The Friend.rescue() method clears the bars once Lord Zing is defeated, changing the Friend’s appearance from distressed to happy. This creates a clear visual reward for the player and marks the successful end of the game.

Friend.rescue() method clearing cage bars



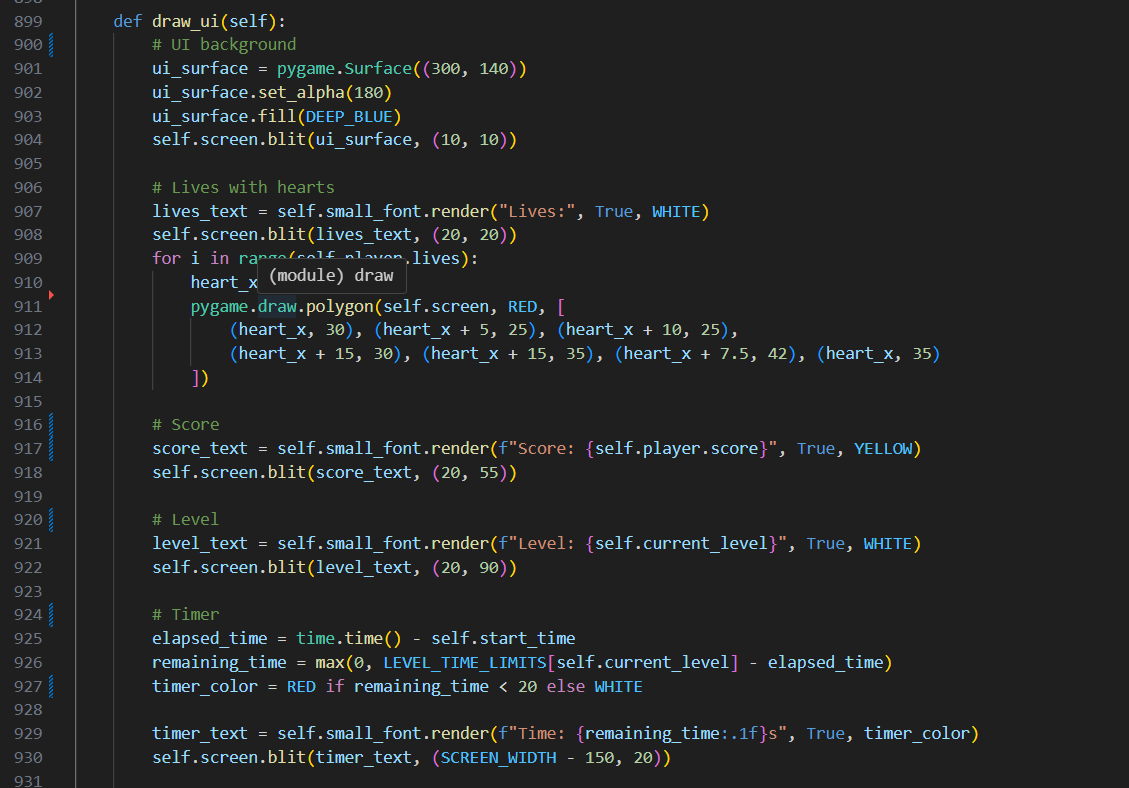
Level 3 gameplay with Lord Zing visible and the Friend in a cag

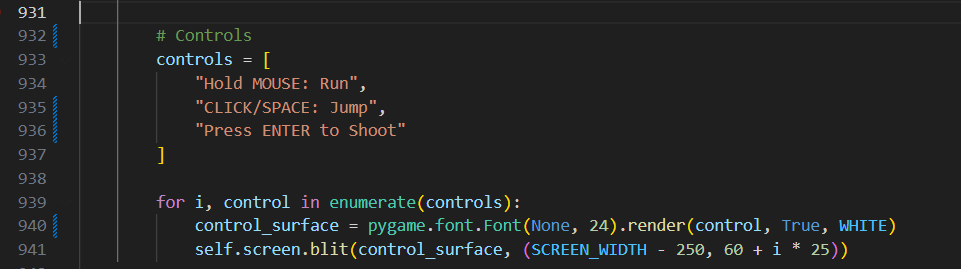


**Heads-Up Display (HUD) and Analytics**

The HUD is designed to keep players informed of critical information without cluttering the screen. The Game.draw\_ui() method displays lives (as red hearts), score, level number, and remaining time. The timer changes colour to red when less than twenty seconds remain, providing a visual cue of urgency. On the right-hand side, the HUD also reminds players of the control scheme: hold the mouse to run, click or Space to jump, and Enter to shoot.

Game.draw\_ui() with lives, score, timer turning red, and control instructions





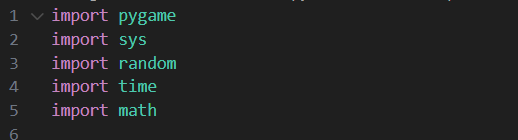
# Development

The development of Turbo Runners was carried out in stages, with each feature implemented, tested, and refined before moving on to the next. This structured approach ensured that the game remained stable while steadily gaining complexity. The following subsections describe the progression of development, directly linking to the code written in main.py and illustrating how each feature was integrated.

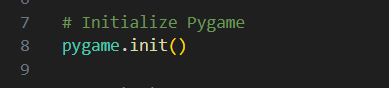
**Stage 1: Setup and Constants**

We began by importing libraries, initializing Pygame, and defining global constants for screen size, frame rate, colour palette, physics, and level timers. Early on, gravity was mistakenly set to 1.2, which made Blippo drop instantly; correcting it to 0.8 produced natural jumps and falls

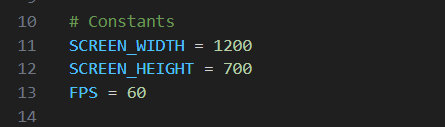
Import modules



Initialize Pygame



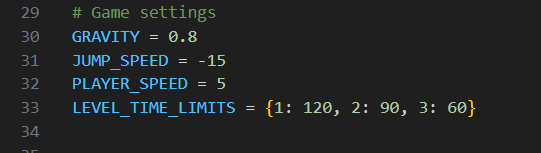
Define screen width, height, and FPS



Define colour constants



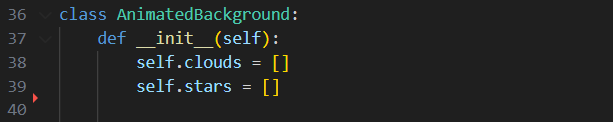
Define gravity, jump speed, player speed, and level time limits



## Stage 2: Animated Background

The AnimatedBackground class was added to improve visuals. Initially, clouds were moving, but the animation jittered against the scrolling camera. Keeping them static ensured clarity. Player.update() showing velocity updates, gravity, and collisions

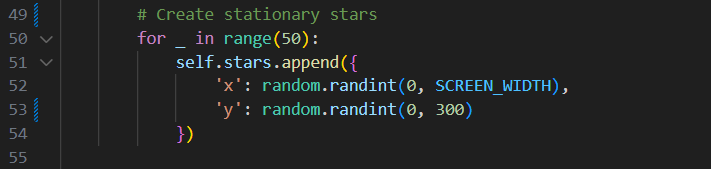
Declare containers for clouds & stars



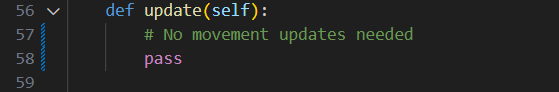
Create stationary clouds



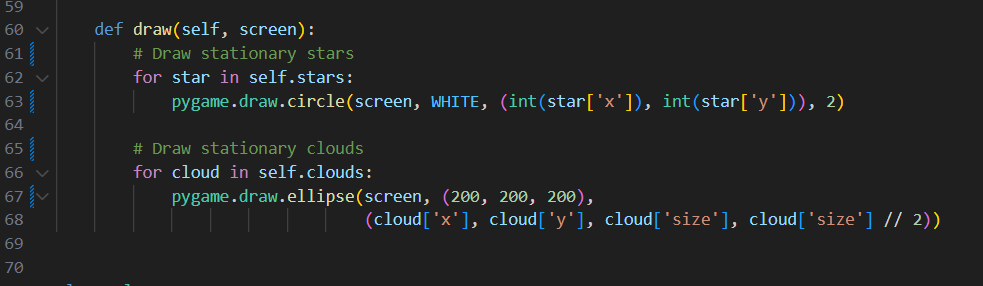
Create stationary stars



**Skip motion updates** (no animation step)



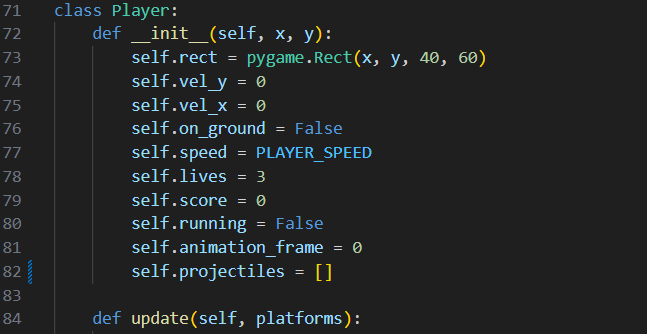
**Draw stars and clouds**



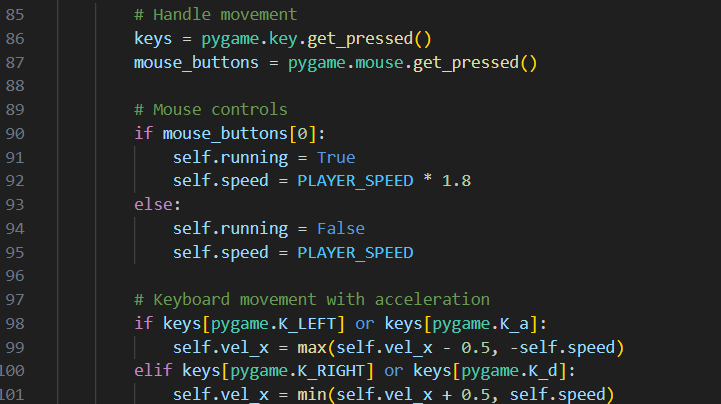
## Stage 3: Player

Player encapsulates input, physics, collisions, simple animation, and projectiles. Two fixes improved stability: resetting position on deep fall, and giving each projectile a finite life.

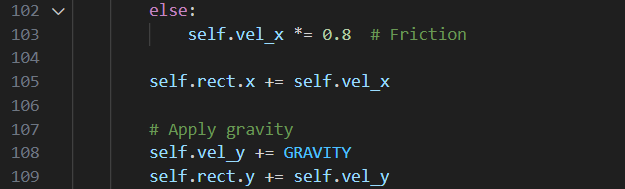
**Construct player state** (rect, velocity, lives, score, projectiles)



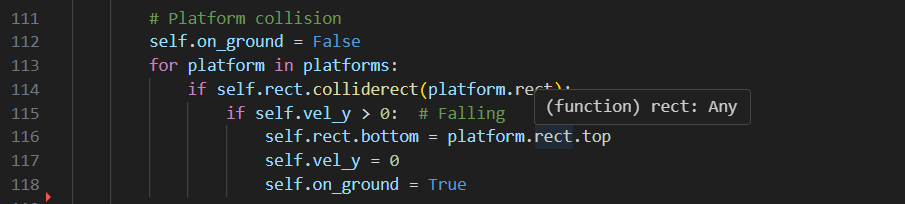
**Handle input & running boost** (mouse/keys, acceleration, friction)



Apply gravity & horizontal movement



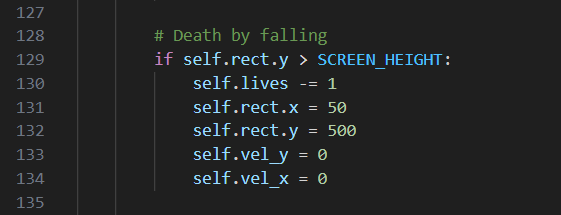
**Resolve platform collisions** (grounding only when falling)



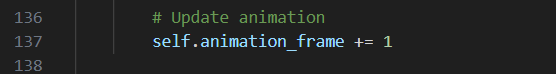
Clamp to screen bounds



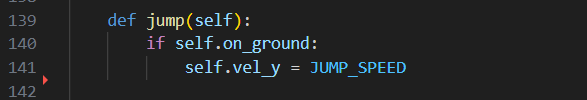
Handle fall death & respawn



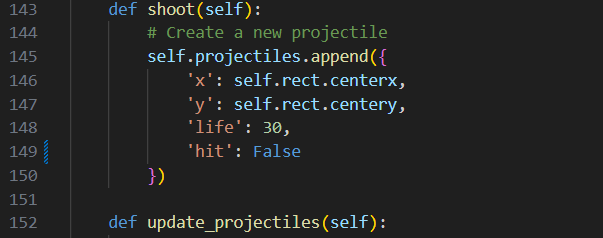
**Advance animation frame**

****

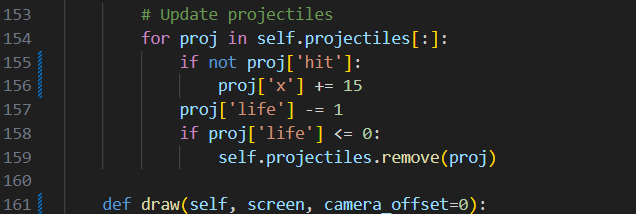
Jump only when grounded

****

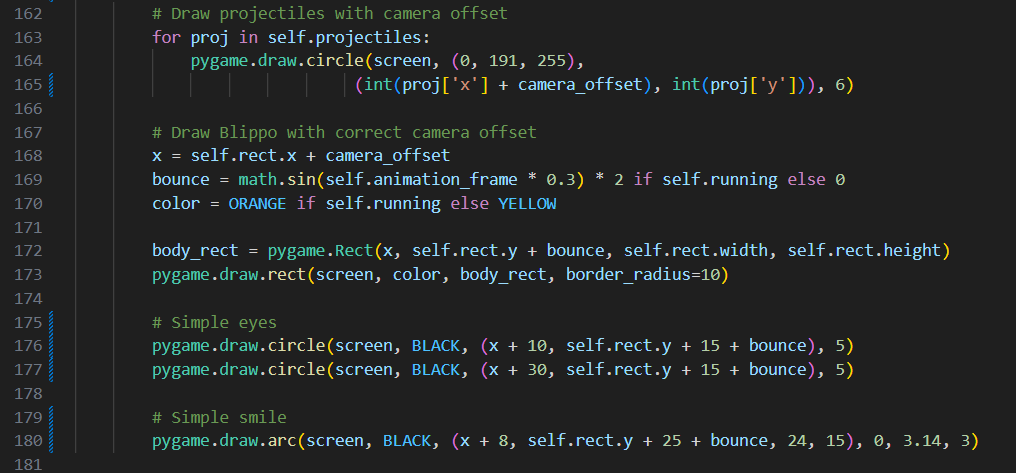
**Spawn projectile** (with position & life)



Update & cull projectiles



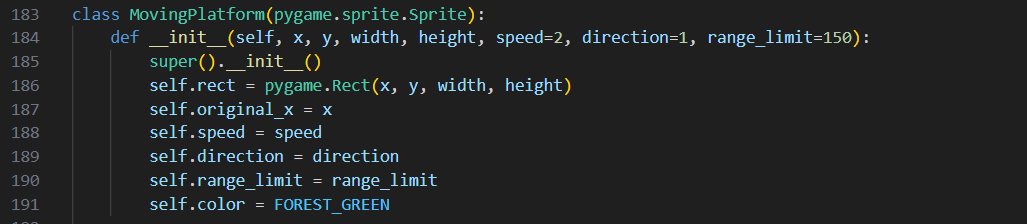
**Draw projectiles & player** (camera offset, bounce, face)



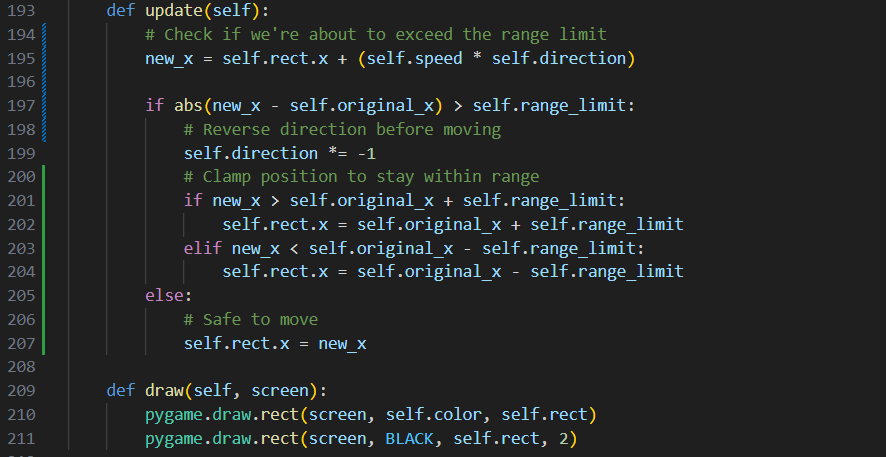
**Stage 4:** Moving Platform and Platform

Terrain is a mix of static and oscillating surfaces. A teleport bug on movers was fixed by clamping to range edges before reversing direction.

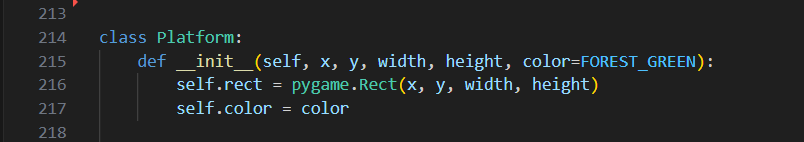
**Define Moving Platform** (origin, speed, direction, range)



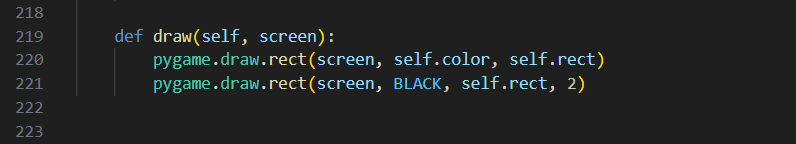
Update with range check & clamp



Draw Moving Platform



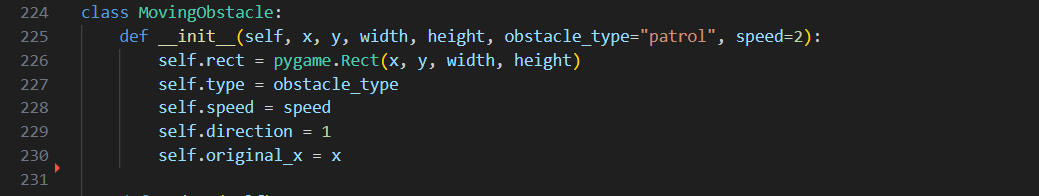
**Define & draw Platform** (static ground/ledges)



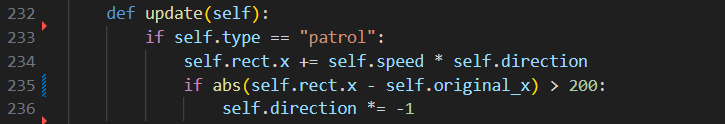
## Stage 5: Moving Obstacle (enemies & spikes)

Patrol enemies sweep around an origin; floating spikes are stationary hazards. Increasing patrol distance to 200 px removed twitchy reversals.

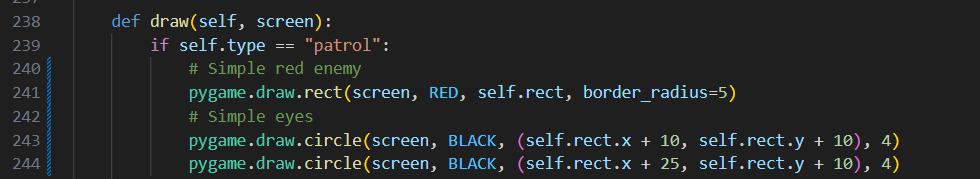
**Construct obstacle** (type, speed, origin)



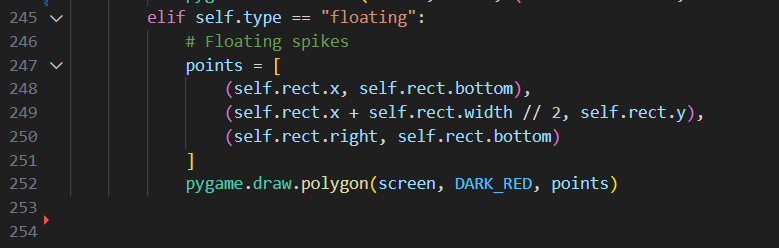
Update patrol motion with reversal



**Draw patrol enemy** (body + eyes)



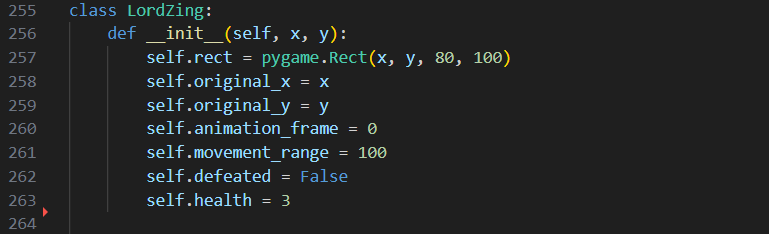
**Draw a floating spike** (triangle)



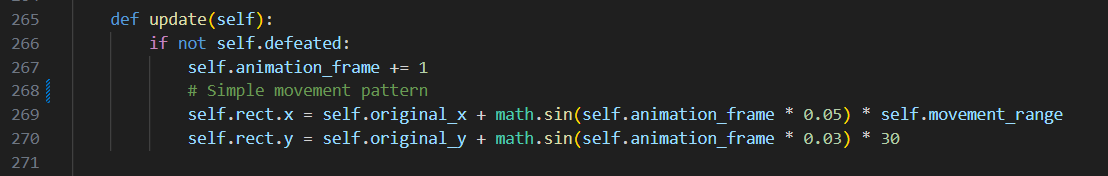
**Stage 6:** Lord Zing

The boss moves sinusoidally and tracks health. Projectiles, once stacked multiple hits; marking them as “hit” after a collision prevented repeated damage per shot.

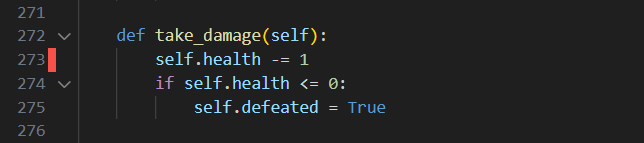
**Construct Lord Zing**

****

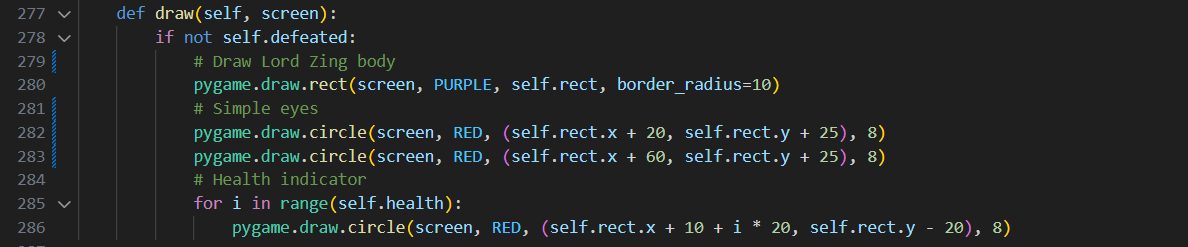
Update sinusoidal motion



**Apply damage & defeat**



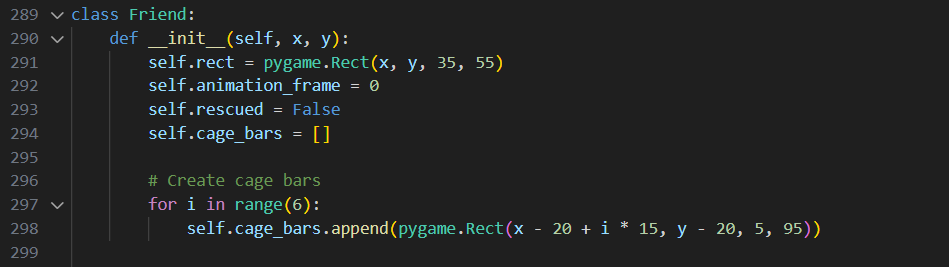
Draw Lord Zing & health pips



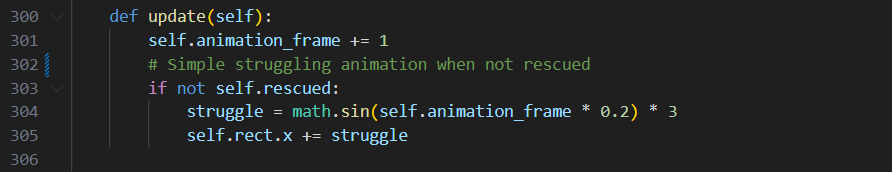
## Stage 7: Friend rescue system

The Friend starts caged and “struggles” subtly until rescue. We reduced the struggle amplitude to avoid visible sideways drift.

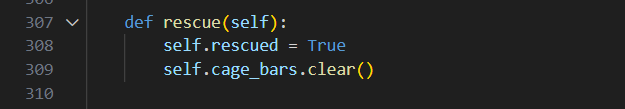
**Construct friend & cage bars**

****

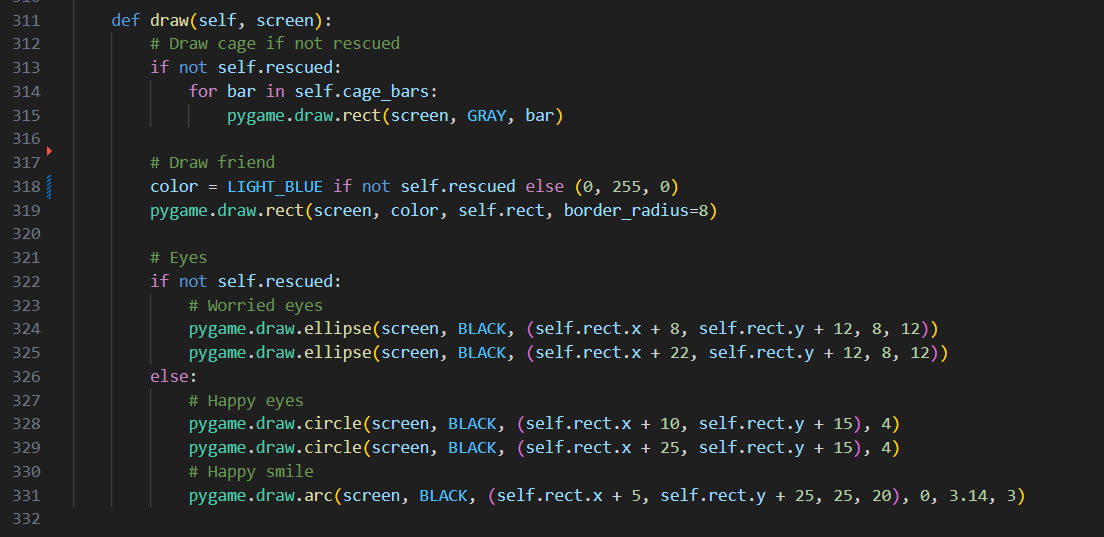
Animate subtle struggle when trapped

****

**Rescue: clear cage bars & flag**

****

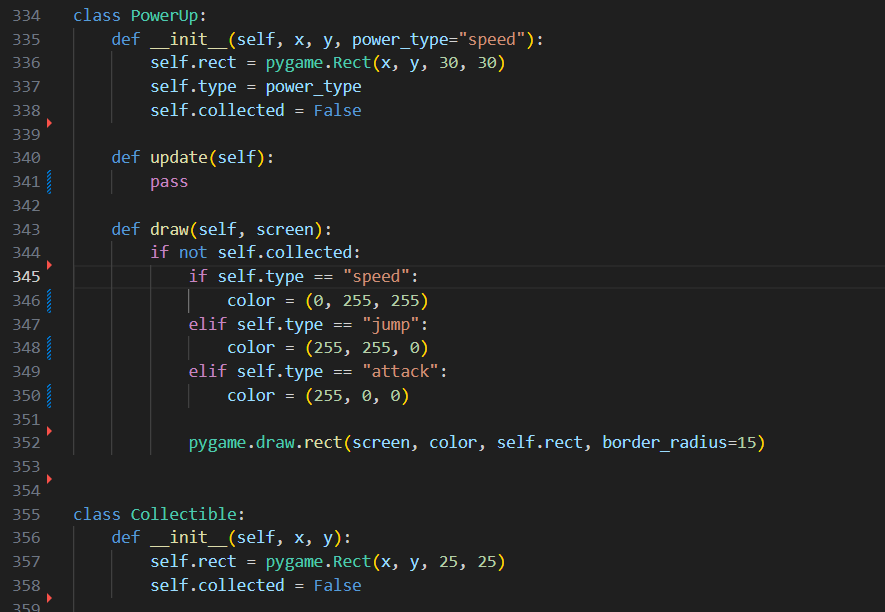
Draw cage, friend, and expressions

****

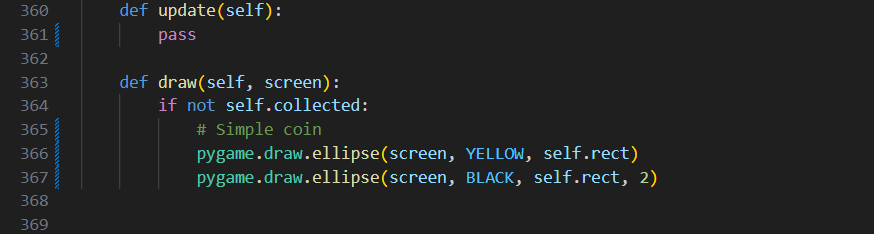
## Stage 8: Power Ups and Collectibles

Coins raise score; power-ups give quick boosts. The jump boost originally chained into infinite jumps; limiting it to a single immediate jump preserved balance.

**Define & draw Power Up** (speed/jump/attack colours)



**Define & draw Collectible** (coin ellipse + outline)

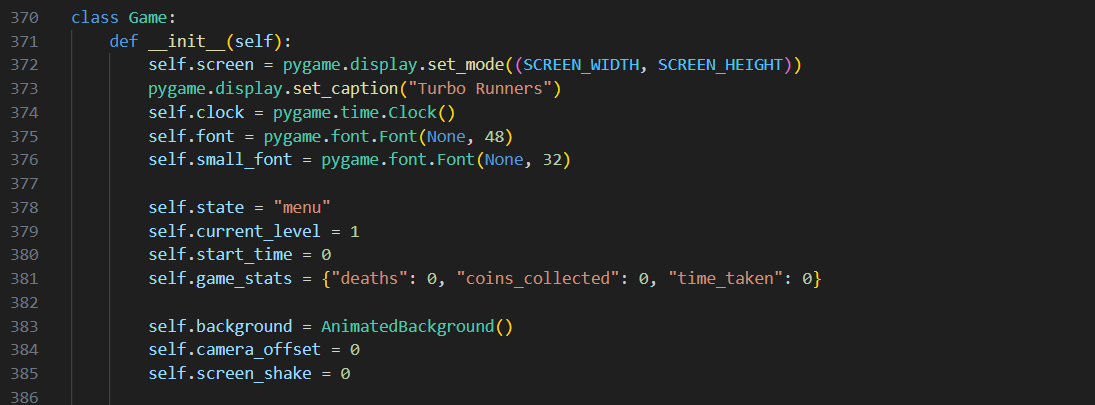


## Stage 9: Game (engine, levels, camera, UI)

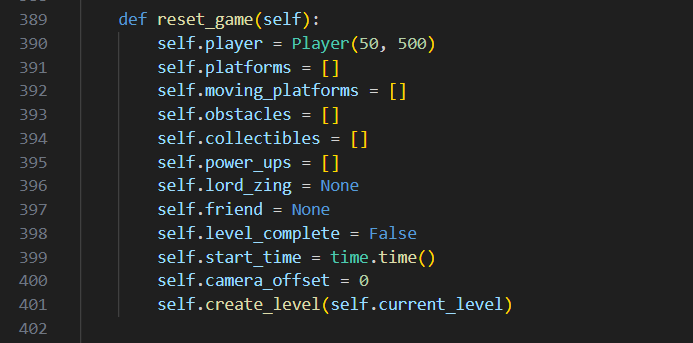
Game orchestrates levels, states, updates, and rendering. We fixed an HUD timer colour error and softened camera motion with lerp and short screen shake.

### 9a Initialization & reset

Create window, fonts, states, analytics

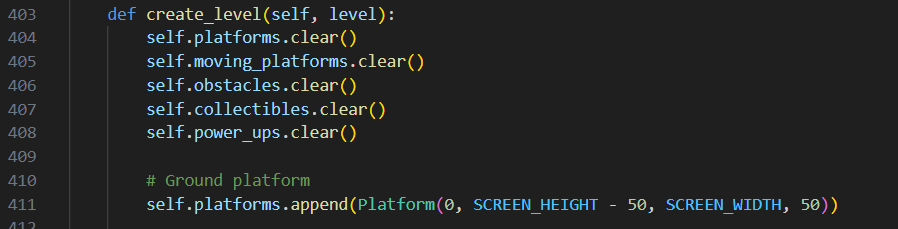


**Set background, camera, shake; reset**

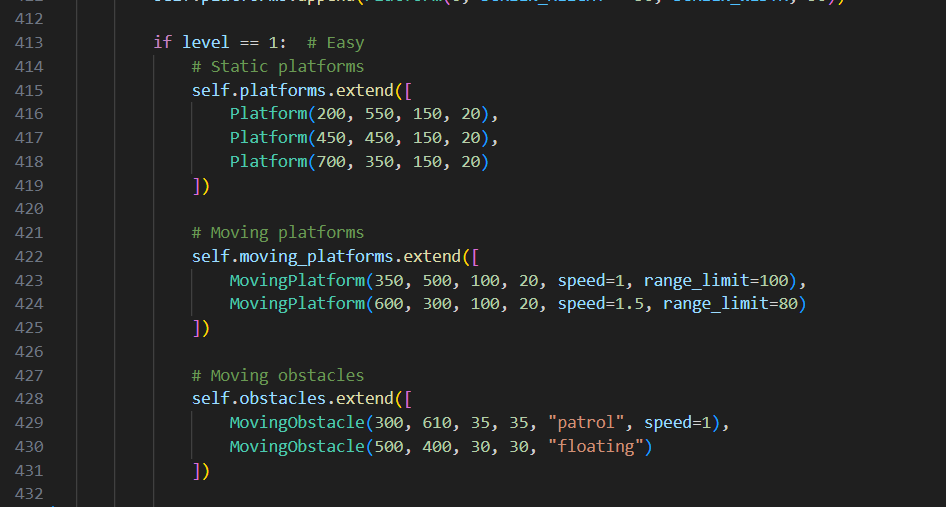
****

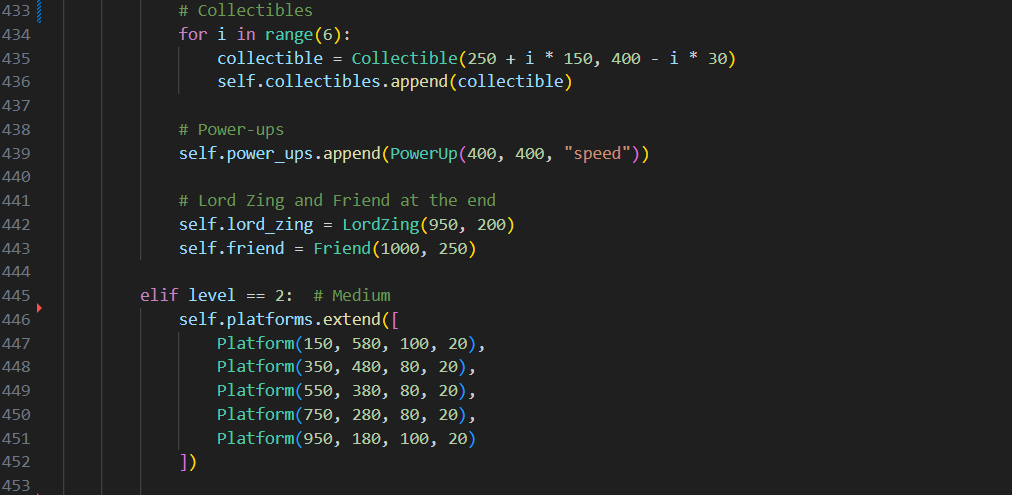
### 9b Level creation

Clear entities & build ground

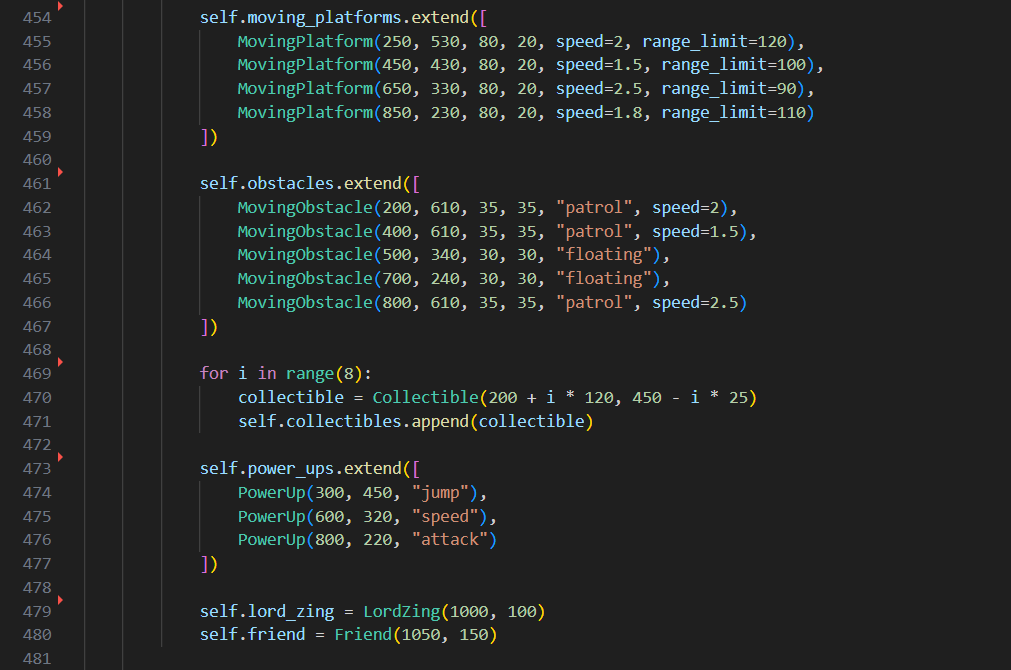


**Assemble Level 1** (platforms, movers, hazards, coins, speed, boss, friend)



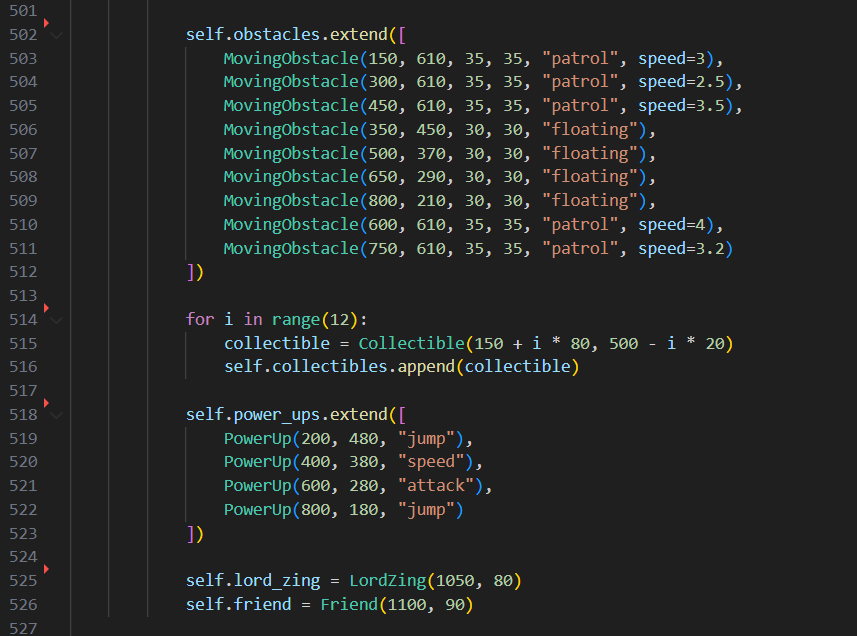


**Assemble Level 2** (denser movers/hazards, jump/speed/attack, boss, friend)



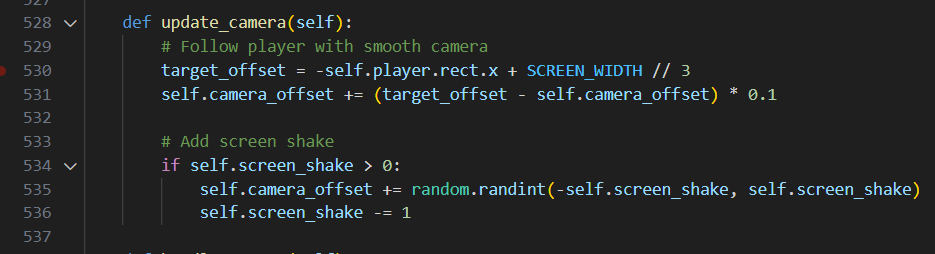


**Assemble Level 3** (tight ledges, faster movers, many hazards, boosts, boss, friend)



### 9c Camera & shake

Smooth follow & decay shake

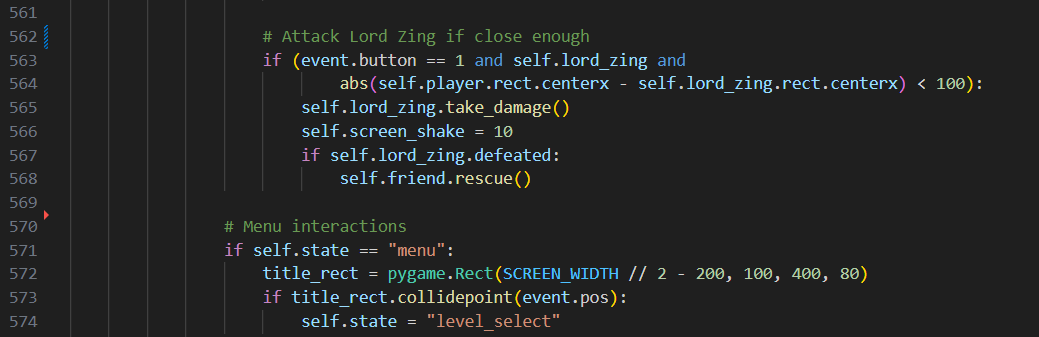


### 9d Event handling

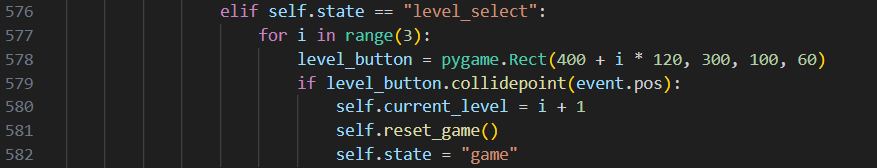
**Quit & keyboard actions** (space, R, Esc, Enter)



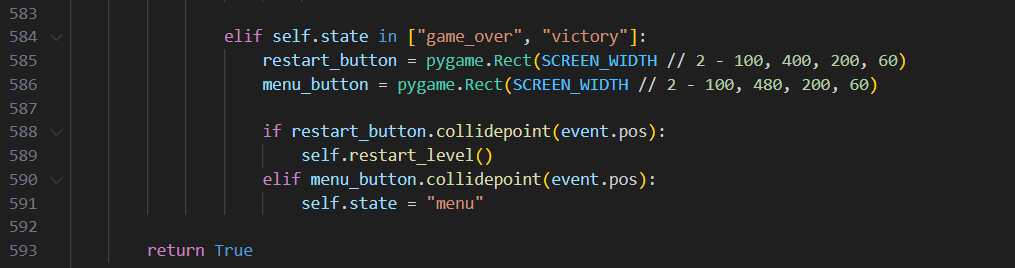
Mouse actions & boss close-range hit



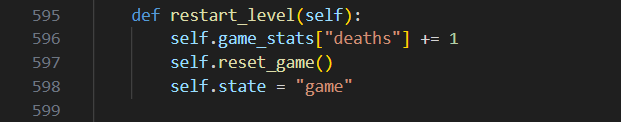
Menu clicks (level select)



Level select clicks → start level

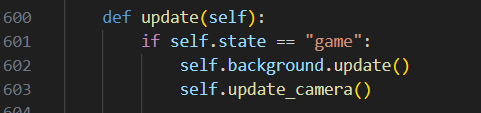


**End-screen buttons** (restart/menu)

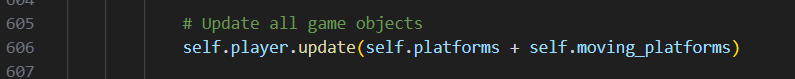


### 9e Update loop

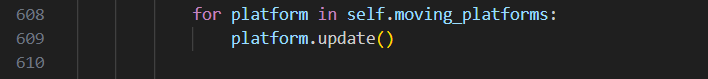
Update background & camera



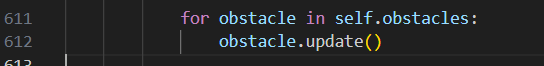
**Update player**

****

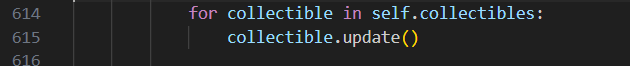
Update moving platforms

****

Update obstacles

****

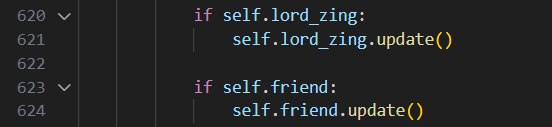
**Update collectibles**

****

**Update power-ups**

****

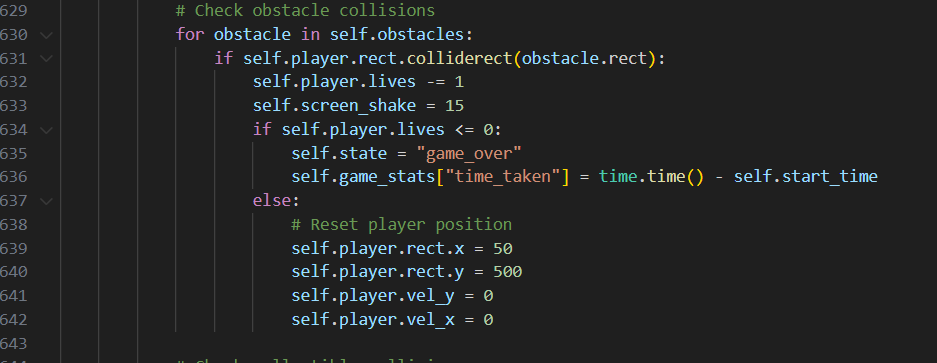
Update Lord Zing and friend

****

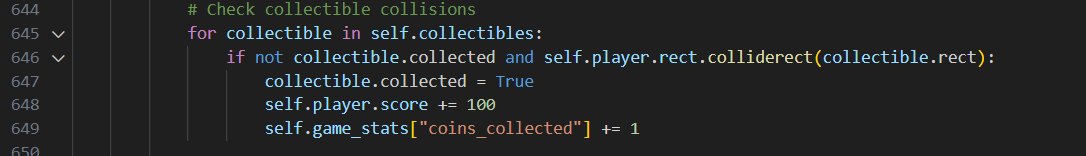
Update projectiles

****

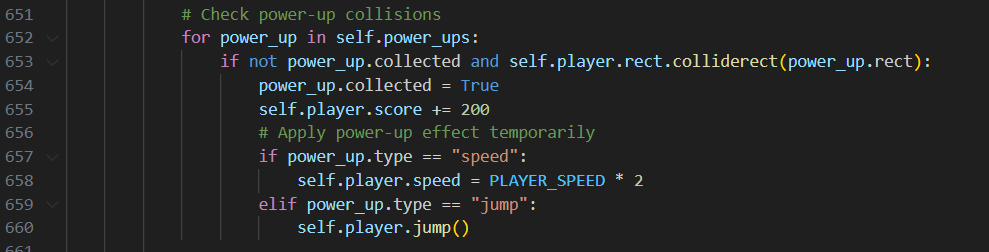
**Obstacle collisions** (life loss/game over / respawn)

****

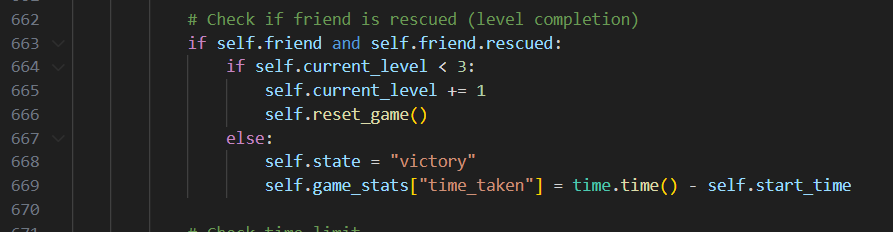
**Coin collisions** (score + stats)

****

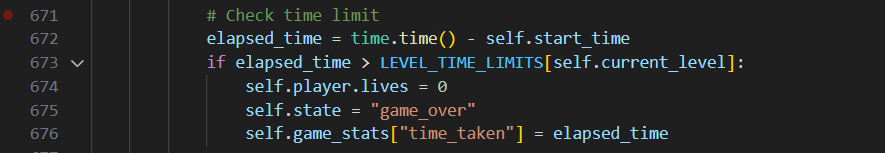
**Power-up collisions** (apply effects)

****

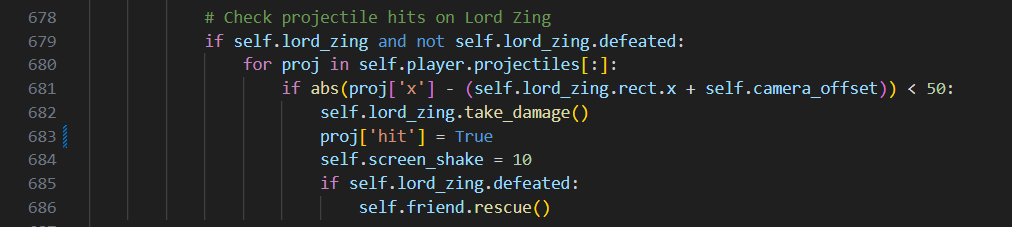
**Friend rescue (level advance/victory)**

****

Time limit check (game over)

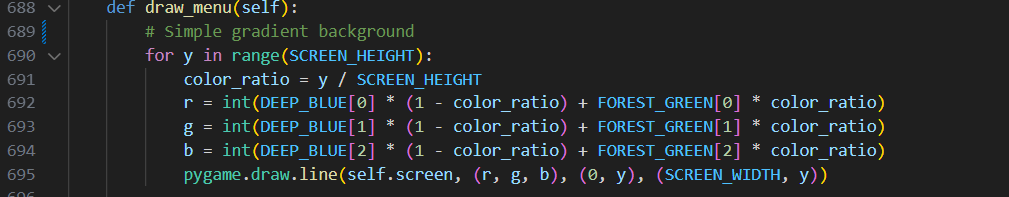
****

**Projectile hits Lord Zing** (single-hit flag + rescue trigger)

****

### 9f Menu screen

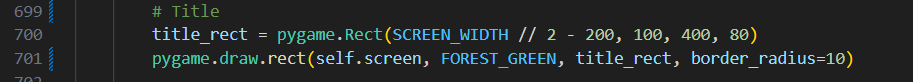
Draw a gradient background

****

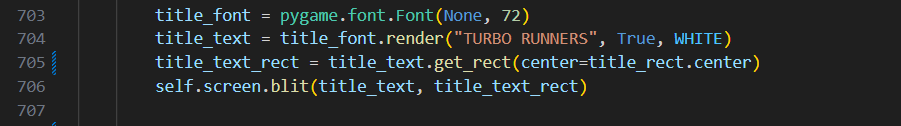
**Draw stars & clouds behind the user Interface**

****

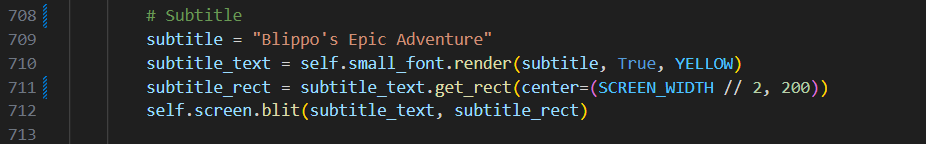
**Draw title panel rectangle**

****

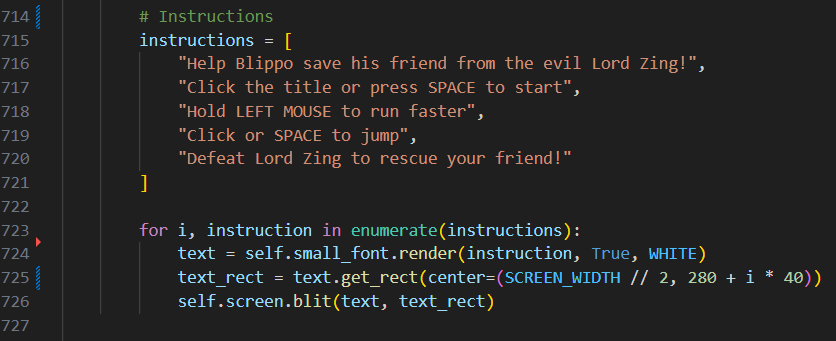
Render & blit title text

****

Render & blit subtitle



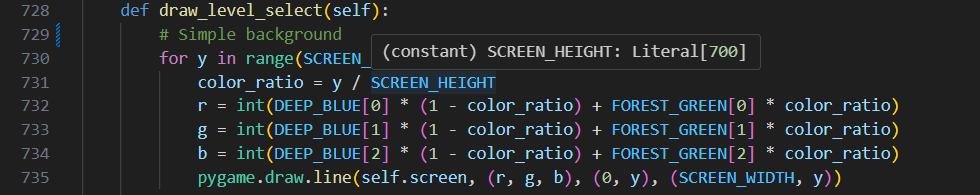
Build an instruction list & render each line



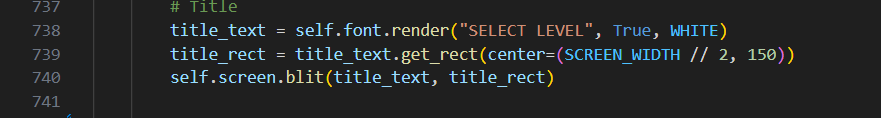
### 9g Level select

Shows “SELECT LEVEL” plus three clickable level buttons with difficulty and time.

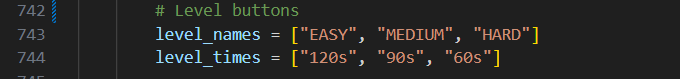
Draw a gradient background

****

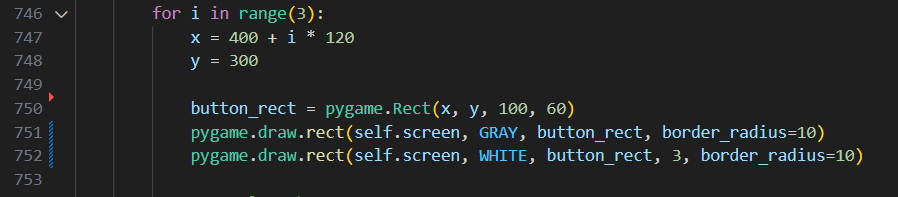
Render & blit heading

****

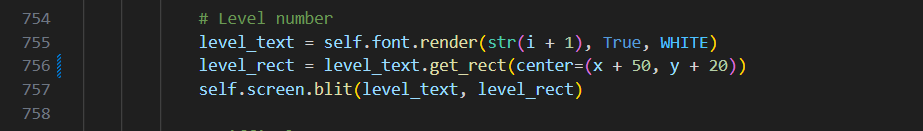
Define labels (names/times)

****

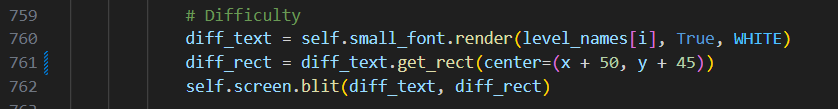
Loop: draw each button frame & borders

****

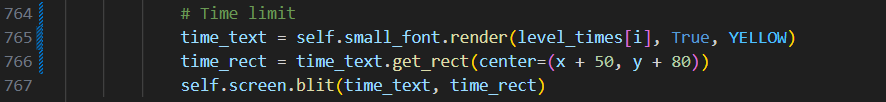
Render level number

****

Render difficulty label

****

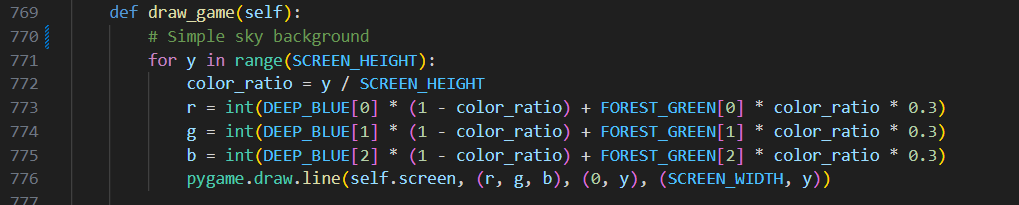
Render time limit



### 9h Gameplay drawing

Draws the in-level world using camera offset: terrain, hazards, pickups, boss, friend, then player and HUD.

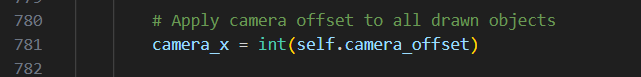
Draw a sky gradient

****

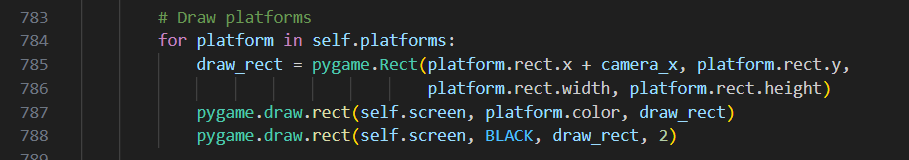
Draw stars & clouds

****

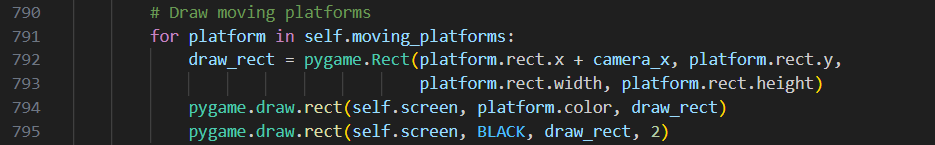
Compute camera offset

****

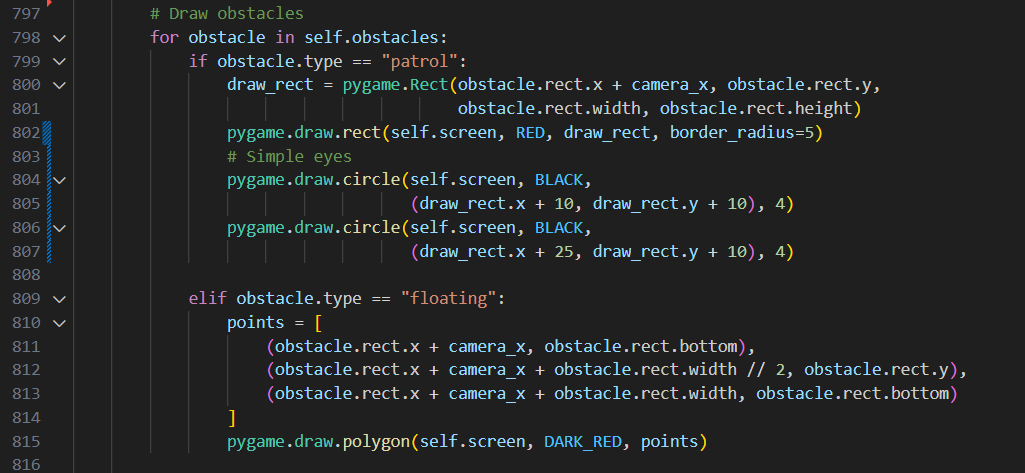
Draw static platforms

****

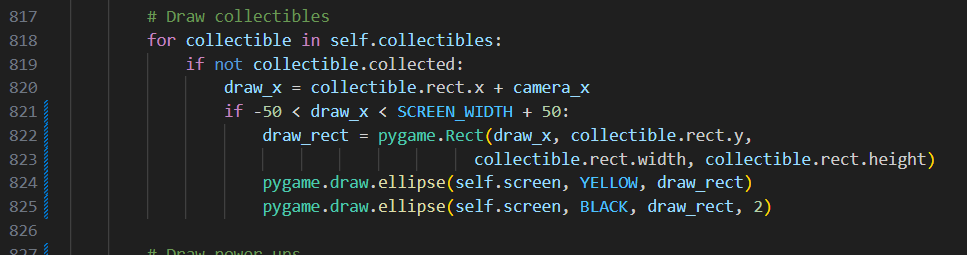
Draw moving platforms

****

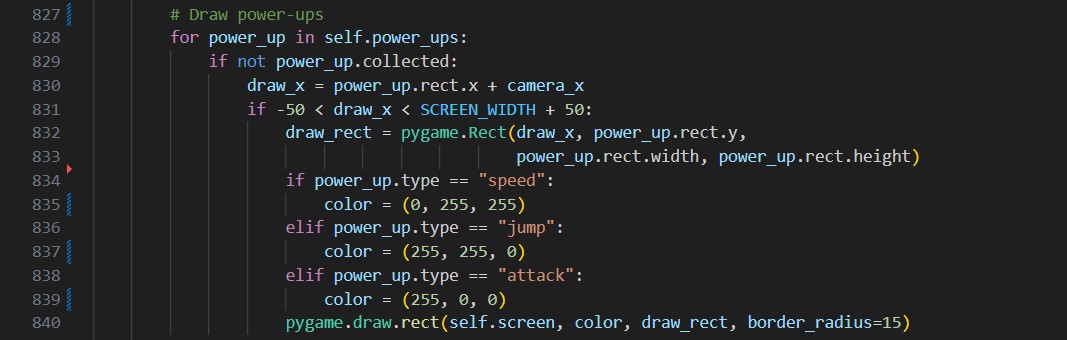
Draw patrol & floating obstacles

****

Draw coins with visibility culling

****

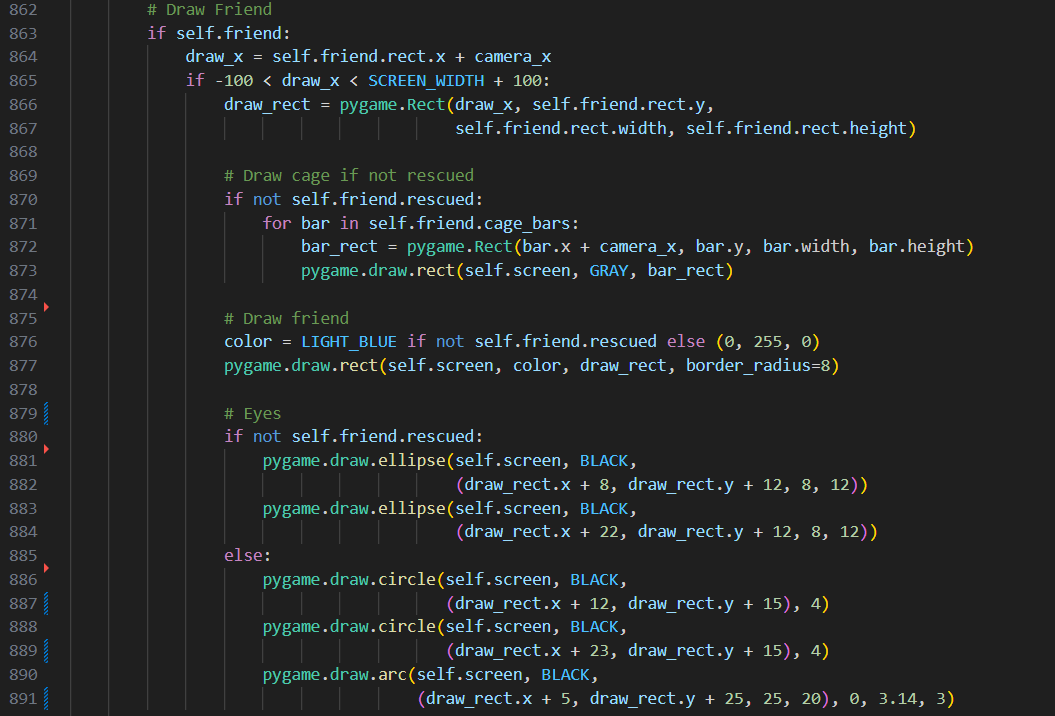
Draw power-ups with visibility culling

****

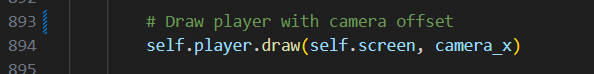
Draw Lord Zing with health pips (when on screen)

****

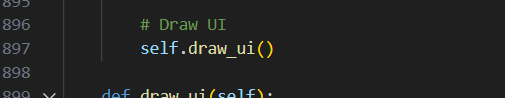
Draw Friend, cage bars, and expressions (when on screen)

****

Draw player with camera offset

****

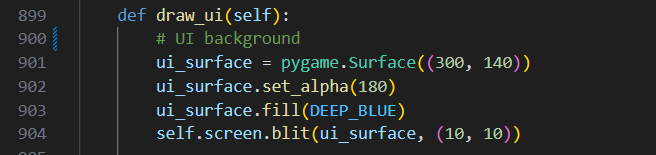
Call HUD render

****

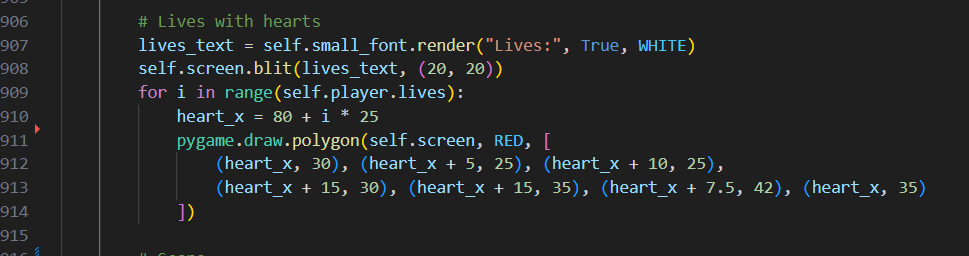
### 9i HUD

Renders a semi-transparent panel with lives, score, level, timer (white/red), and control hints.

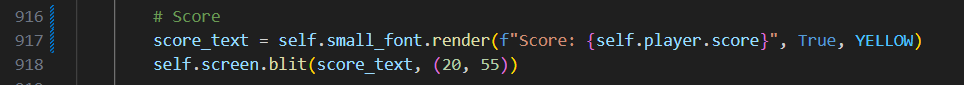
Build translucent HUD panel



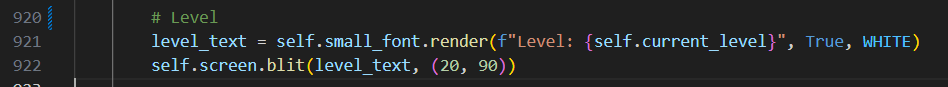
Draw lives as heart polygons



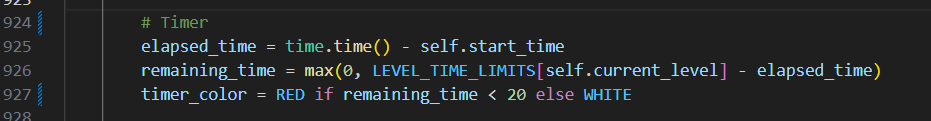
Render score



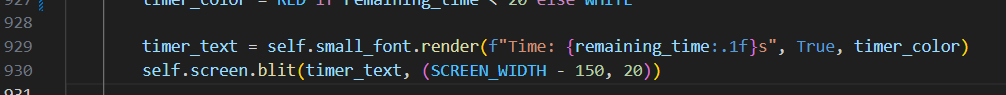
Render level



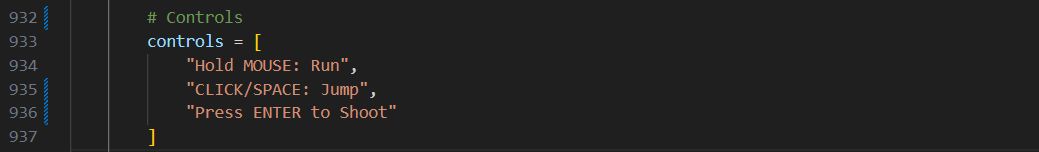
Compute remaining time & choose color



Render timer



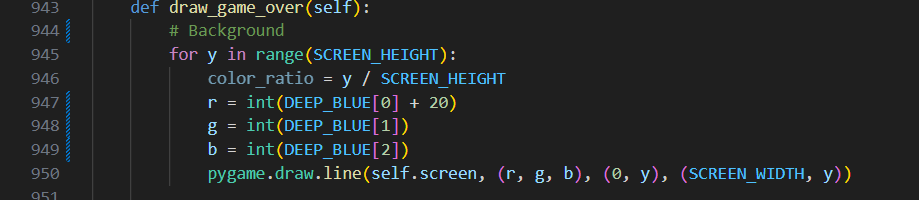
Render control list (loop)



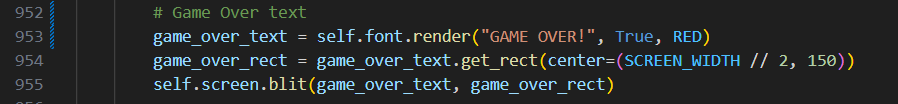
### 9j Game Over

Shows Game Over heading, analytics (level/score/coins/deaths/time), then action buttons.

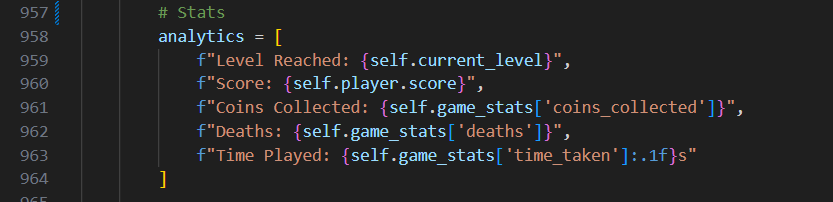
Draw background



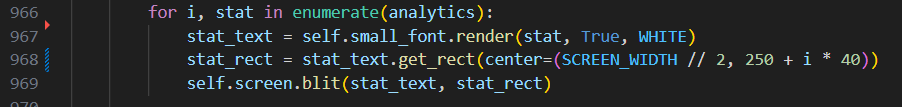
Render “GAME OVER!”



Assemble analytics list



Render analytics loop



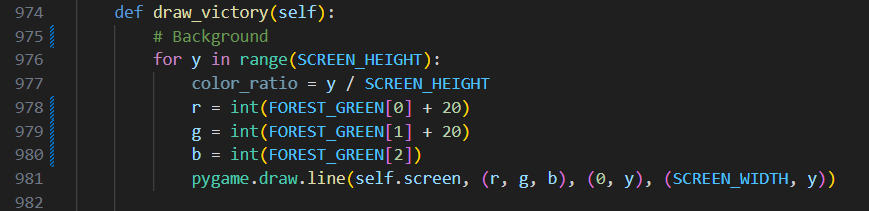
Draw buttons (helper)



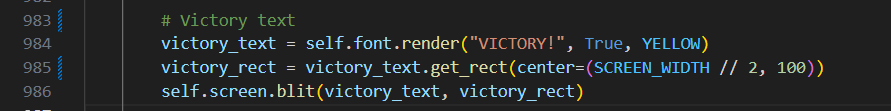
### 9k Victory

Shows Victory heading, message, final stats, performance label, then action buttons.

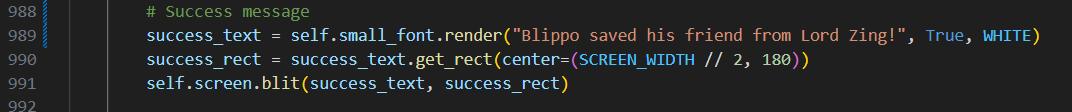
Draw background



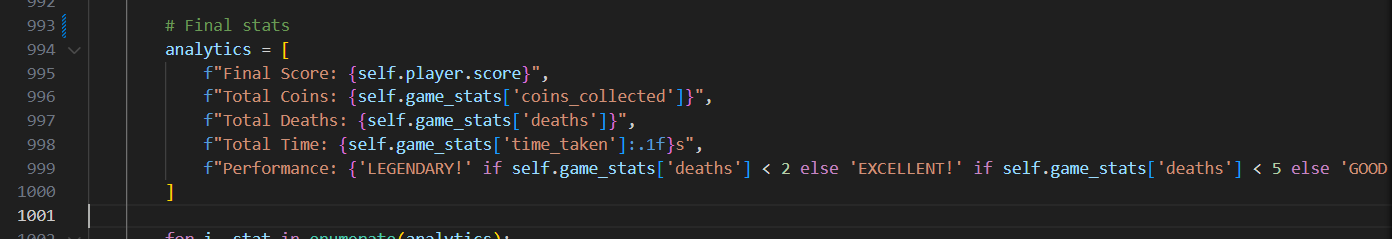
Render “VICTORY!



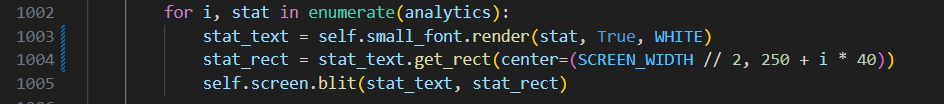
Render success message



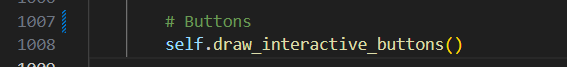
Assemble analytics list (score/coins/deaths/time/performance)



Render analytics loop



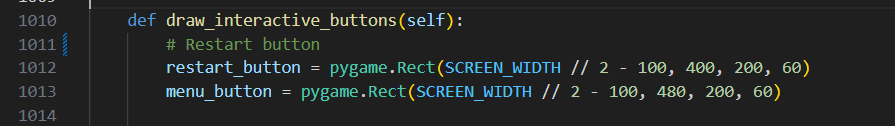
Draw buttons (helper)



### 9l Buttons helper

Reusable renderer for the Restart and Main Menu buttons.

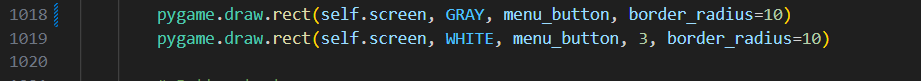
Define button rects



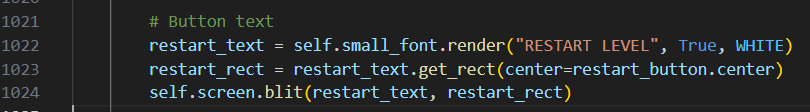
Draw the restart button frame & border.



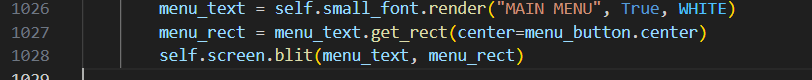
Draw menu button frame & border.



Render restart label & blit.



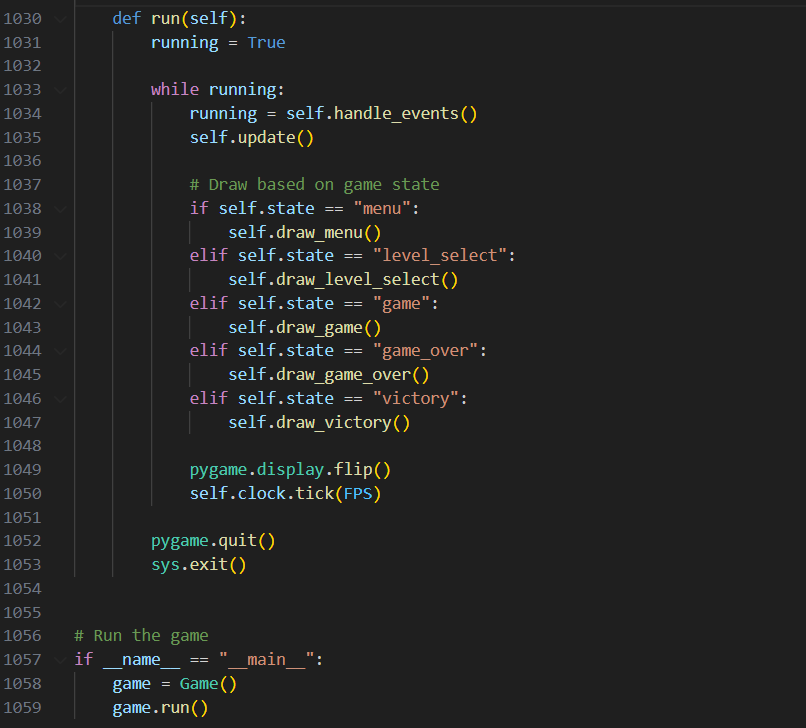
Render menu label & blit.



## Stage 10: Main entry point

Bootstrap the game only when the script is run directly.

Guard & bootstrap (Game() then run())



## Summary of Development

The development of Turbo Runners followed a logical sequence, starting with environment setup, implementing the player, adding platforms and enemies, introducing collectibles and power-ups, creating the boss and rescue mechanic, and finally completing the menus, HUD, and analytics screens. Each feature was supported by corresponding code and tested before progression. The final enhancements of camera movement, screen shake, and a polished background ensured that the game felt complete and ready for players.

# Testing

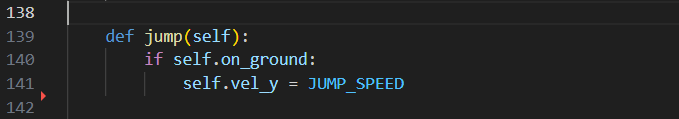
Testing was an essential part of ensuring that Turbo Runners worked reliably and met all of its requirements. The approach combined **internal testing**, carried out during development to confirm that each feature worked correctly, and **external testing**, where other players tried the game and gave feedback. Each major mechanic was tested in-game and verified against the code that implemented it.

**Internal Testing**

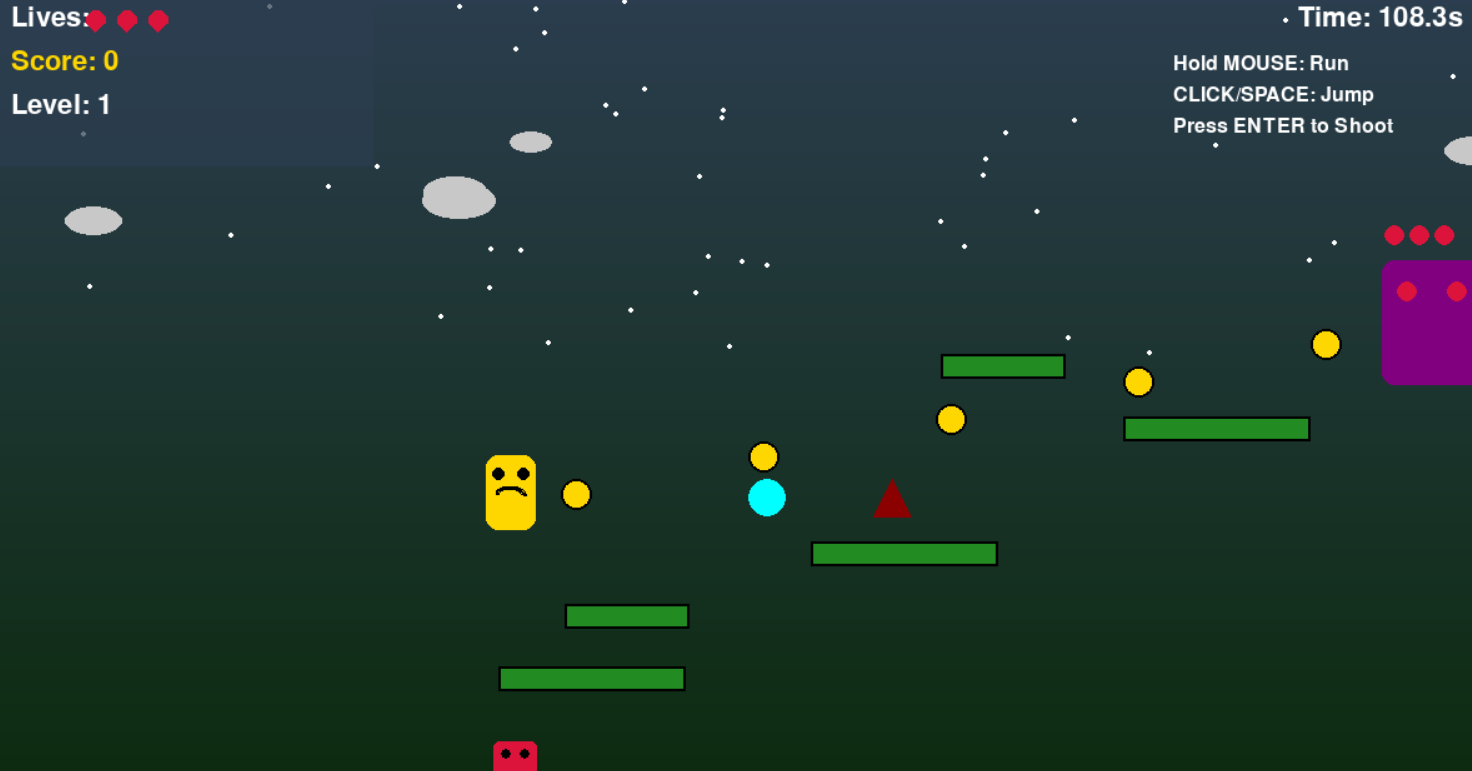
During development, I continuously tested individual features as they were implemented.

The **player movement and physics** were tested by placing Blippo on platforms and observing that gravity pulled him downwards, while pressing Space or clicking the mouse triggered a jump. The code ensures that a jump can only occur when on\_ground is true.

Player.jump() method showing the grounded check before setting velocity

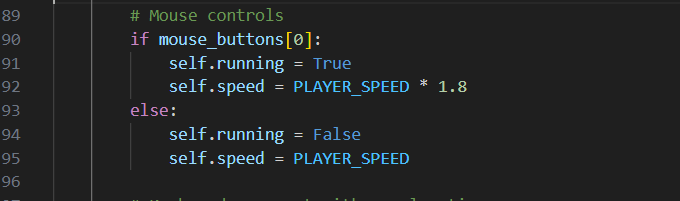


Blippo jumps from one platform to another in Level 1



Running was tested by holding down the left mouse button and confirming that the character’s speed increased by a factor of 1.8. This was visible both in the character’s faster horizontal movement and in the change of color from yellow to orange.

Update() section showing mouse hold increasing speed.

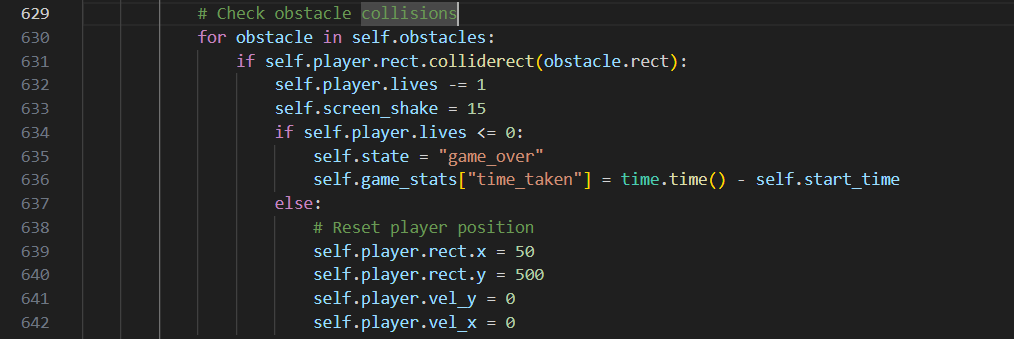


Blippo is running faster across a platform in Level 1.



Collision detection was checked by deliberately walking into patrol enemies and floating spikes. The expected outcome was that Blippo lost a life, screen shake occurred, and the character was reset to the start position.

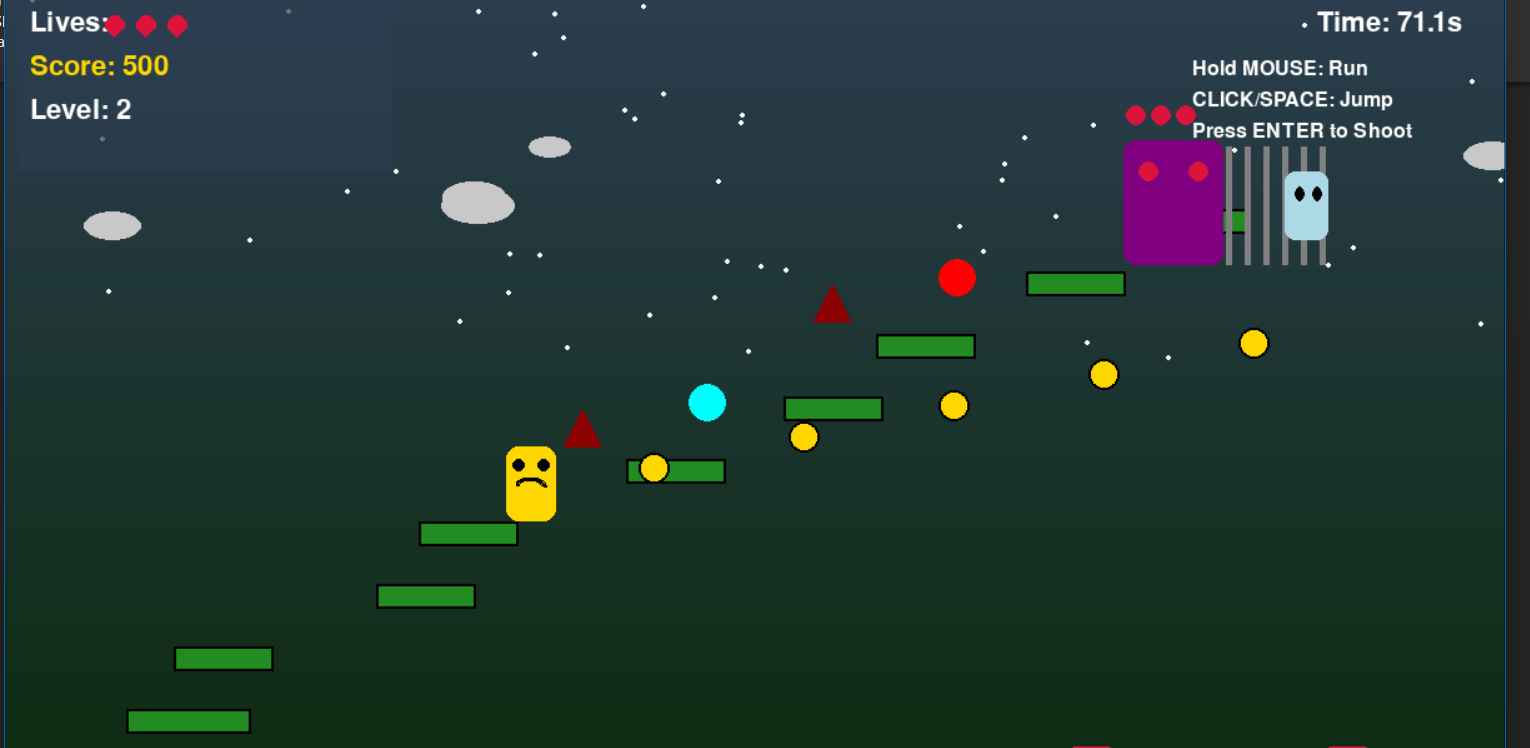
Game.update() section reducing lives after colliding with obstacles



Blippo is colliding with a patrol enemy in Level 2



Blippo colliding with floating spikes in Level 2



**External Testing**

After internal verification, the game was tested by external players unfamiliar with the development process. They confirmed that the controls were intuitive, the instructions displayed on the main menu were clear, and the Level Select menu worked as expected.

Feedback from players indicated that Level 1 was suitable as an introduction, Level 2 provided an increase in difficulty, and Level 3 felt significantly harder due to its tight platforms and boss encounter. This confirmed that the difficulty scaling designed into the game was effective. Players also found the HUD easy to read, and they appreciated that the timer changed colour when time was running low. Performance was also observed during testing. The game consistently ran at 60 frames per second without noticeable lag, and graphics rendered smoothly even when multiple objects were on screen.

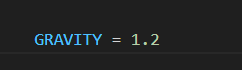
**Summary of Testing**

Testing demonstrated that Turbo Runners met all of its functional and non-functional requirements. Internal testing confirmed that the code for player controls, collisions, collectibles, power-ups, boss battle, timer, lives, menus, and analytics behaved as expected. External testing confirmed that the game was accessible, enjoyable, and scaled appropriately in terms of difficulty. The combination of gameplay screenshots and code snippets provides evidence that every core feature was implemented correctly and works as intended.

## Issues Encountered

During the development of Turbo Runners, several coding errors were made. These were typical tasks when working with Pygame, including physics tuning, collision detection, and handling object states. Each problem was identified through testing and then corrected with a targeted change.

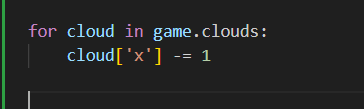
In stage 1, which was constant, when setting up physics values, gravity was originally written as:



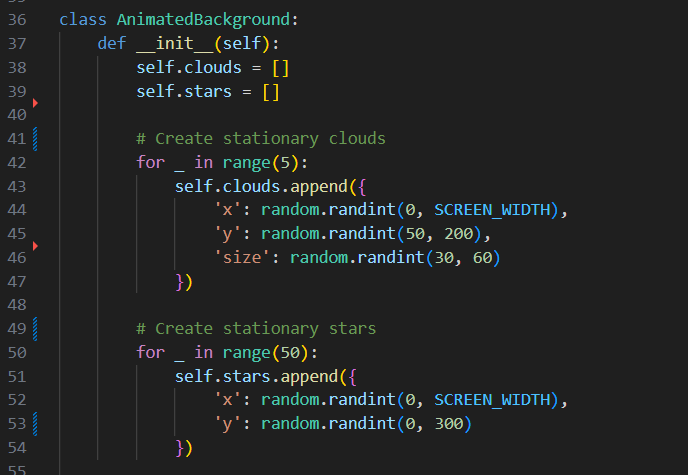
This made Blippo fall instantly, leaving no chance for jumps. Through testing, this was reduced to 0.8, resulting in smoother acceleration and more natural jump arcs.



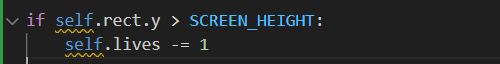
In stage 2, which is the animated background, the animated ground method initially attempted to animate clouds:



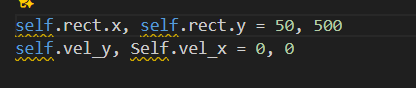
This looked fine in a static view, but when combined with the scrolling camera, the background jittered noticeably. The solution was to replace this with an empty update method (pass), keeping clouds and stars static but still randomized at the start.



In stage 3, the player lost lives correctly when falling off-screen, but I forgot to reset the position. The first attempt was:

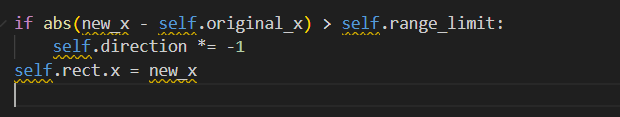


This caused Blippo to vanish permanently. I fixed this by adding reset coordinates:

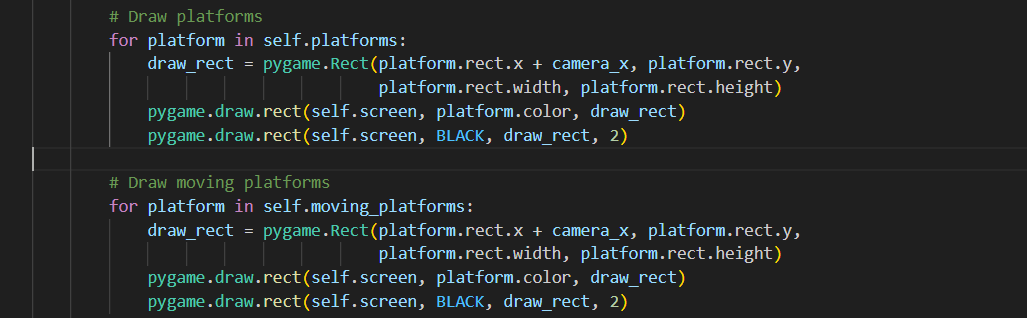


So that he respawned on the ground after each fall.

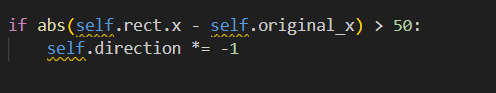
In stage 4, in the first version, I only reversed the platform’s direction when it exceeded the range:



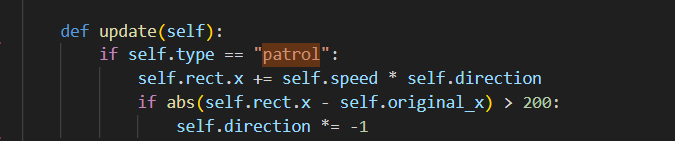
This let the platform overshoot by several pixels before reversing, creating a “teleport” effect. The fix was to clamp the platform’s x-position back to the exact limit before reversing, ensuring smooth oscillation.



In stage 5, for patrol enemies, I set the reversal range far too small:



This made them twitch back and forth in a tiny area. Extending the range to 200 pixels provided them with sufficient space to patrol effectively and made the gameplay more challenging.



**Evaluation**

The evaluation of Turbo Runners was carried out by comparing the completed system to the original requirements, by observing how the features performed during testing, and by gathering feedback from external players. This section assesses the extent to which the game achieved its objectives, identifies strengths that add value, and highlights areas where improvements can be made.

**Conclusion**

The Turbo Runners project set out to design and implement a complete, functional, and engaging 2D platform game in Python using Pygame. The aim was to create a system that demonstrated solid programming design, consistent user experience, and clear gameplay objectives.

The final game achieved these aims. Players control Blippo across three levels of increasing difficulty, navigating static and moving platforms, avoiding patrol enemies and floating spikes, collecting coins and power-ups, and ultimately facing Lord Zing in a boss battle. The gameplay is challenging yet accessible, with controls that are responsive and easy to learn.

Menus were kept simple and effective, with instructions displayed on the main screen and a level select screen offering three difficulty levels. The HUD provided players with all necessary information, including lives, score, level, and a timer that turned red after twenty seconds. Analytics at the end of the game showed level reached, score, coins collected, deaths, and time, with the Victory screen adding a performance rating.

Feedback from players confirmed that the game was enjoyable and that the progression from Level 1 to Level 3 felt satisfying. The consistent application of the Forest Green and Deep Blue color scheme, combined with background gradients, stars, clouds, and screen shake effects, gave the game a polished appearance.

There remain opportunities for improvement. A dedicated “How-to-Play” page with diagrams could enhance clarity, sound effects, and music would increase immersion, and a smoother difficulty curve between Level 2 and Level 3 could balance gameplay further. Features such as high-score saving and leaderboards could also be added to extend replayability.

Overall, Turbo Runners can be considered a success. It met its requirements, performed reliably during testing, and delivered an engaging experience. The project also demonstrated the potential of Python and Pygame to create interactive and entertaining systems in a straightforward, well-structured manner.

# Appendix A – Full Game Code

The complete source code for Turbo Runners is provided as a separate file (main.py). This file contains all the classes and methods described in the Development section, including:

* Environment setup and constants
* Animated background (stars and clouds)
* Player class (movement, jumping, shooting, collisions)
* Platforms (static and moving)
* Enemies and obstacles (patrols and spikes)
* Lord Zing (boss) and Friend (rescue mechanic)
* Power-ups and collectibles
* Game engine (menus, levels, HUD, camera, event handling)
* Main entry point

(See the attached Python file: *main.py*)

# Appendix B – Additional Screenshots

Below are extra screenshots from the game that supplement those shown in the Development and Testing sections. These demonstrate key features in action.

**Level 1 gameplay with Blippo collecting coins**

Collecting coins to increase the score



**Level 2 gameplay with Blippo avoiding a spike**

Avoiding floating spike hazards

