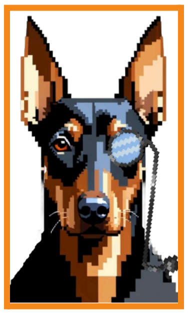
CFG Degree Autumn 2024

Dogtective Documentation

Abbeygayle, Estelle, Iman, Melanie, Sally, Zarrin



1. Introduction

In this report we outline the design and execution of our game 'Dogtective'. We cover the implementation of this using the Pygame library in Python and database management in SQL.

1.1. Project Overview

The Dogtective game is a 2D, retro arcade inspired game, where the player uses intuitive controls (i.e. arrow keys) to control our protagonist. The premise is as follows: Our protagonist, the Dogtective, takes on a case to find his client's lost toy.

1.2. Purpose and Scope

The game aims to be an enjoyable, lighthearted escape with the initial intention of appealing to a younger pre-teen audience. Our game also has the capacity to appeal to more casual gamers of any demographic.
The development of the game focused on creating one main level, with the option to expand further.

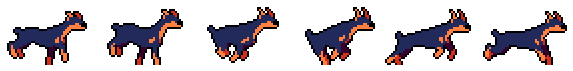
2. Project Background

2.1. Inspiration

It was clear from initial group discussions that we had a shared love for animals hence we wanted to have an animal protagonist. We took inspiration from games such as Crossy Roads and Sonic for gameplay and design. We wanted to incorporate an intuitive user interface with high replayability since the player will be encouraged to beat their high score.

2.2. Core Mechanics

It is a 2D game where our protagonist needs to dodge obstacles (cars) to reach the end of the level. The player starts at the main menu, where they have the option to select 'Play', 'Leaderboard', or 'Credits' by clicking on the relevant button. If they click 'Play', the window will change to the gameplay screen, where the player will use their arrow keys to control the protagonist. Navigation to the leaderboard from the main menu shows the game's top ten scores and the credits show all those involved in the project.



As our protagonist will need to dodge obstacles, total health will start at five bones, and will deplete when they collide with obstacles in the game. The player will receive a ‘Game Over’ Or ‘Mission Complete’ screen depending on the outcome of their playthrough. Both of these screens also have a ‘Play Again’ button.

Our level is also timed. This is important, as we reward the player with bonus points if their playthrough is under 30 seconds. Points are calculated based on the amount of health left multiplied by one thousand, plus any bonus points acquired. The ‘Mission Complete’ screen displays lives left at the end of the playthrough and points gained.

3. Specifications and Design

3.1. Functional Requirements

1. The game must have **intuitive controls** so the player can maneuver the character out of the way of obstacles and progress the level.
2. A player must receive their **score** at the end of the level, which can then be viewed in the leaderboard menu.
3. The game must return a **‘Game over’ or ‘Mission Complete’** screen, depending on the playthrough.

3.2. Non-Functional Requirements

1. There is the possibility to **scale up** the game by adding more levels and incorporating other game mechanics (e.g. puzzles, side scrolling background etc.).
2. The game is **responsive** to user input, with minimal delay from the player pressing arrow keys to move the character, or the player pressing a button on screen to navigate the menu .

3.3. Architecture

The diagram below (*Figure 1*) shows how our system works, with a second diagram (*Figure 2*) to set out how scores are stored and received from the database (DB).

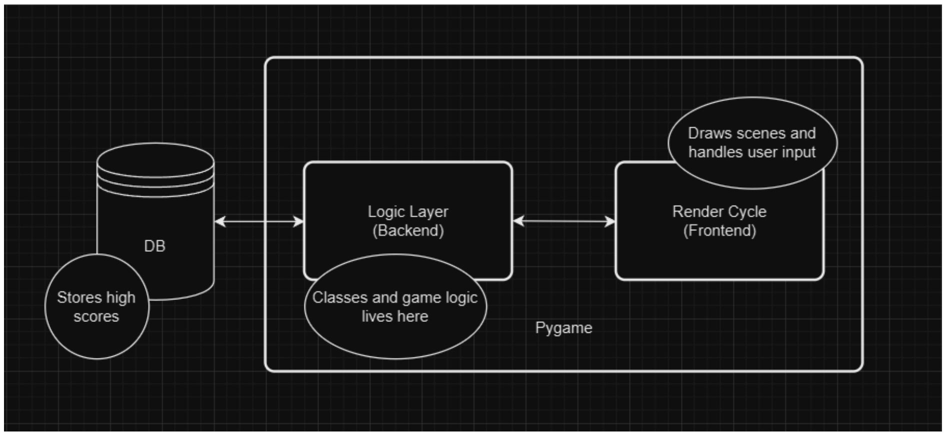


Figure 1: Overview of project architecture

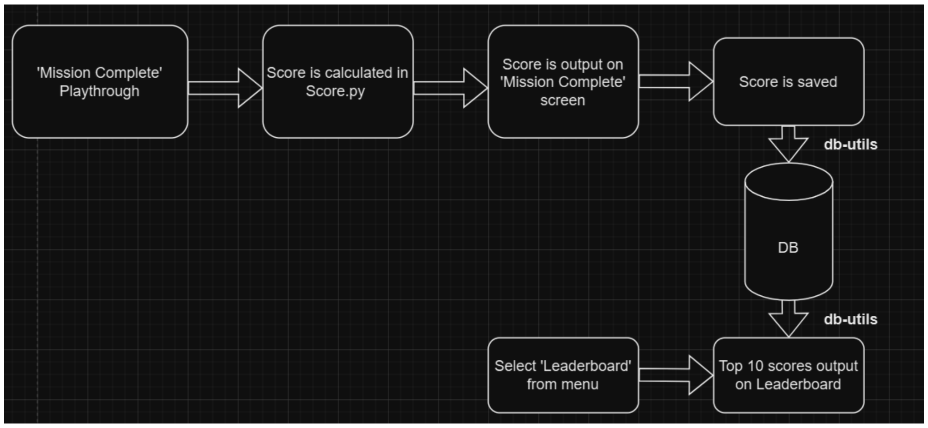

'Mission Complete' Playthrough
Score is calculated in Score.py
Score is output on 'Mission Complete' screen
Score is saved
db-utils
DB
db-utils
Select 'Leaderboard' from menu
Top 10 scores output on Leaderboard

Figure 2: In-depth look at how the game interacts with the database to show high scores on the leaderboard

3.4. Gameplay Mechanics

Using arrow keys, the player is able to navigate the character through the various obstacles to reach the lost toy. The user can navigate to different screens from the main menu by clicking on the relevant button. The user can navigate back to the main menu by using the ‘Menu’ button.

3.5. UI/UX Design

3.5.1. Asset management

Using the Trello board, we assigned ourselves to find open source images to push to the asset folder within the logic folder of the git repository. The images could then be used within Python to animate sprites and show images on screen.

Emphasis was put on finding pixel art and fonts that were vibrant to mirror our vision of an arcade style game. When looking for fonts, we were mindful to find an accessible and readable font for the user as our target was a younger audience. We had envisioned a doberman as the protagonist, and were able to source sprite sheets and icons for free to realise this. Sources for assets used in the game are listed in the references.

3.5.2. Sound design

Background music and sound effects were custom designed for our game, courtesy of Ahmed Abdi, with the dog, detective and arcade-style game themes in mind. These have been incorporated into the game to improve immersion and experience for the user. The game features sound effects for damage taken. These give more impact to the interactions within the game.

4. Implementation and Execution

4.1. Development Approach and Agile Methodology

Through agile methodology we used the MoSCoW analysis to decide the initial concept and as a process on what to work on first - the must haves, to then work on the should haves and if time, work on the could haves. Team members were assigned different roles such as project leads, testing, UI/UX and more. We used Trello tickets to set out the MoSCoW analysis and to show each team member's assigned, outstanding and in-progress tasks, which were relevant to the role they were assigned. We used Git branches to work on features concurrently before peer reviewing pull requests and merging into the main branch. We also refactored, where appropriate, following reviews.

Furthermore, separation of components and features of the game via an ordered file structure and different Python files allowed for a more streamlined and iterative development approach. As a team, it meant we could each work on different features with minimal merge conflicts. It was also easier to distribute work.

We had regular meet-ups using Google Meets and our main communication was via Slack. We had a project workflow, to ensure we followed the timeline we set out for ourselves (see *Figure 3*). We also used Google Drive for storing shared files to work collaboratively.

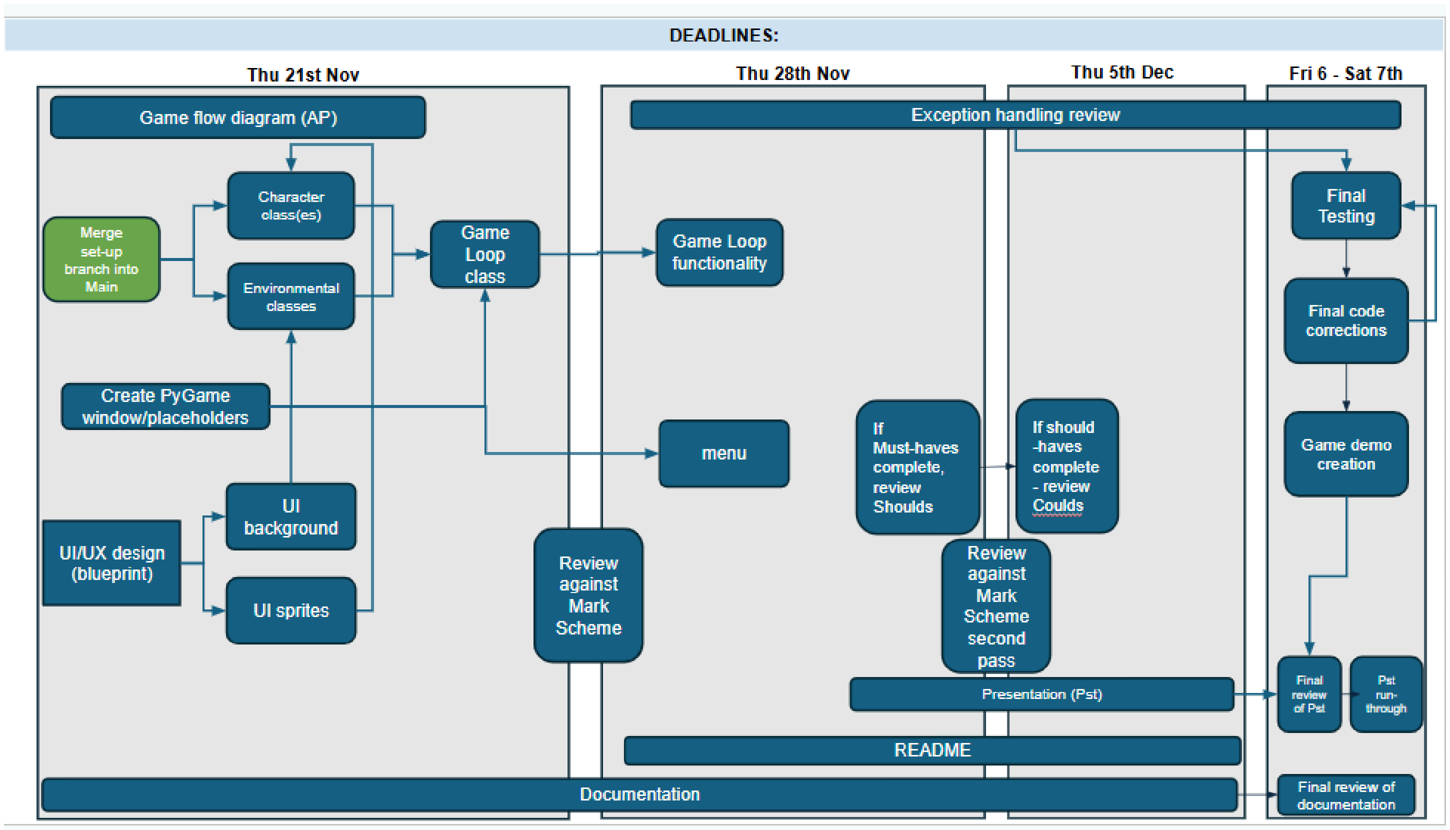


Figure 3: Project workflow

4.2. Implementation

4.2.1. Tools

- The **Pygame library** was needed to implement the game logic
- **SQL** was needed to store and retrieve user scores
- **Inkscape** was used to adapt open source assets for the game
- **Itch.io** was used to source the majority of open source assets

4.2.2. Achievements

There were many achievements for us as a team within this project. However, some of the milestones are:

☑ Object motion: We were able to animate the main character moving with user input. We were also able to have the cars move within the boundaries of the screen at varying speeds.

☑ Collaboration: The team worked collaboratively to navigate the complexities of implementing our idea, with regular team meetings and code reviews to build our game, overcoming many challenges along the way.

☑ Simplifying game run: The game runs only through the game_runner file.

☑ Collision Detection: Added collision detection to decrease health when the dog connects to the car.

☑ Audio: Added background music and sound effects to the game.

☑ Leaderboard: We were able to connect the game to the SQL database to display scores in the Leaderboard menu.

4.2.3. Challenges

Some of the challenges we had throughout the project were:

➔ Vision: What we wanted from the project initially was different from what we could achieve within the short timeframe we had.

➔ Time Management: Members had different schedules and time commitment. Early in the project we lost a member, and another could not join until a couple of weeks into the project.

➔ New Tools: Learning to use the Pygame library for the first time, therefore needing to quickly form a basic understanding to meet targets the team had set out.

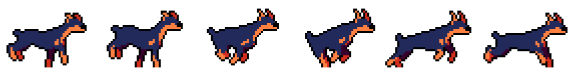
➔ Communication: Adapting to different communication styles and ways of working while using Git collaboratively for the first time.

When faced with challenges, we would come together as a team to decide the best way to approach the challenge and what our next steps were.

4.2.4. Adjustments

Our team had regular meetings and reviews to ensure we stayed on track, and ensure we completed the project within the time frame. Some adjustments we made are listed below:

- Scrolling background: We decided to instead use a static background that contained more obstacles with varying speeds and damage.
- Puzzles: We had initially planned to have puzzles (a ‘should have’ aspect of the game) between each level or that would have been triggered by certain events. This would have been to attract a wider audience.



- Changing character: We wanted the player to have the option to choose their character (a ‘could have’ aspect of the game) but could not add this within the given timeframe so instead had the player be the Dogtective.

4.2.5. Repository Organisation

We have separated the components needed for our game into different folders and files to ensure more readable code that is easier to run. How we did this is as follows:

- We have a ‘logic’ and ‘user interface’ folders to handle different elements of the game.
    - The logic folder contains:
        - Assets (audio, images, font)
        - Components, i.e. classes that allow the assets to be used within the game
        - Database connection
    - The user interface folder contains:
        - The different screens (game, leaderboard, credits, end screen)
        - Runner files for the menu and the game, where the game runner file is where the user would run the game from.
        - Loader files for text and images
        - Game config file
- Our ReadMe file outlines how the game is set up and run by a user
- Requirements.txt tells the user what to install in order to run the game

4.3. Execution

We primarily used object oriented programming to build our game. A class based approach allowed us to more easily integrate different features into the game.

4.3.1. Error Handling and Debugging

We used print statements to solve issues that did not show error messages in the terminal. An example of this would be for events timed in milliseconds, as it meant we had to adjust values to account for this. For error messages that appeared, we researched to see what the errors meant and how to resolve them.

We also used breakpoints and the debugger to see where errors occurred and understand why code did not run as expected. We also used breakpoints within the edges of the code to pause it while it ran to see where the errors occurred, and why code did not run as expected.

5. Testing and Evaluation

5.1. Manual Testing

We manually tested the game using the Gherkin framework. If it did not do what we expected, we would debug to find out why and test again. Most of the testing was visual in terms of seeing what was displayed on the screen and interacting with it. We also carried out regression testing to ensure all previous features were not affected by the addition of any new feature.

Furthermore, we interacted with the game to see what actions caused it to crash, and were able to ask friends to help with testing.

5.2. Functional and Unit Testing

Within database testing we ensured the DB environment was running and using the 'db utils' file we loaded the scores stored within the DB onto the leaderboard screen. Within the SQL file we already had two scores on the file so this is what we were expecting to output on the leaderboard screen.

6. Conclusion

Overall, we are extremely proud of our game. We were able to successfully create an arcade style game using the Pygame library. If we had more time for this project, there are other elements we would have liked to incorporate to further showcase and push forward our skillset. These include:

- Allowing the player to pick their character
- Using math.random to spawn the toy in a random location within a certain area in the level
- Adapting to another platform (e.g. mobile game)
- Creating different levels with different toys, background images, and obstacles
- Incorporating puzzles into the game to add health
- Have animation for hovering over menu button items

This has inspired us to continue our learning journey and growth beyond this course and we are very grateful to CFG for this opportunity

7. References

- Pygame documentation
- Crossy roads initial game ideas
- Bone image
- Doberman Sprite
- Monocle
- Road3 from FreePik
- Street animal asset pack
- City background start menu