CODE FIRST GIRLS

CFG Degree - Autumn Cohort - 2024

THESIS QUEST

A COLLEGE LIFE SIMULATOR RPG GAME

TEAM MEMBERS

ANTIGONY POLLARD
EMILY BUNCE
EMMA BEGUM
GRACE PARR
GWEN MICHAILIDIS
INES DUARTE
JOHANNA MANJA GROENING

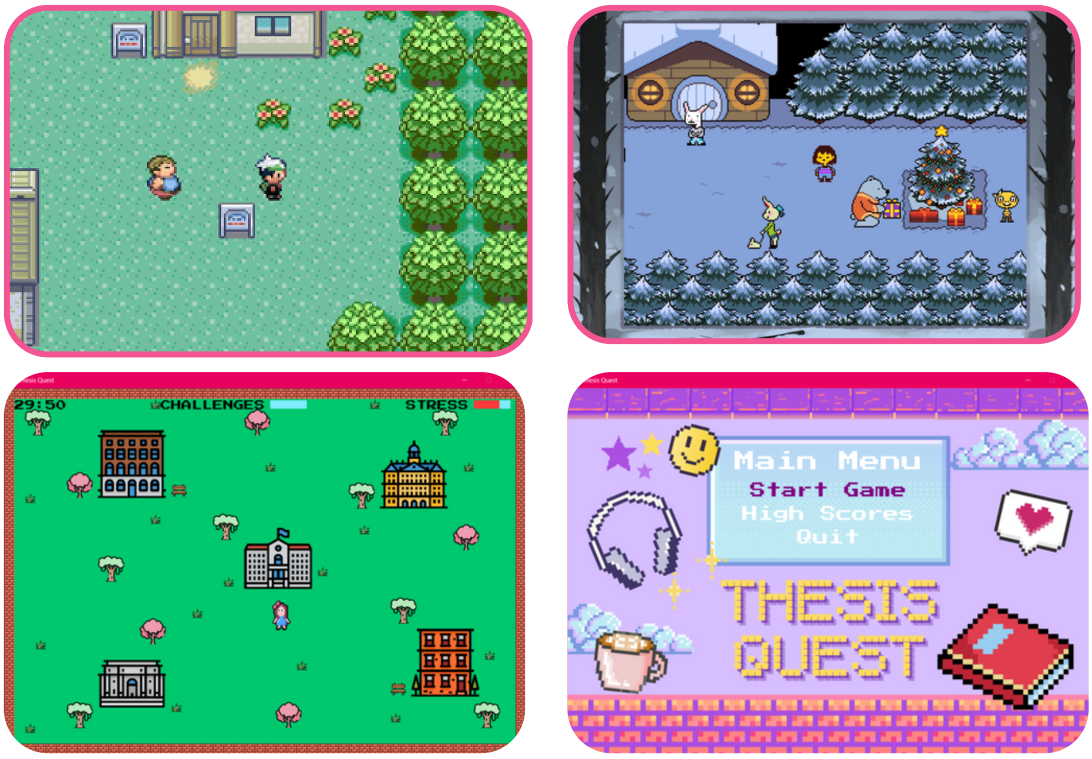
GROUP 1 PROJECT DOCUMENTATION

Code First Girls

December 2024

INTRODUCTION

Thesis Quest emerged from our team's shared vision to create a game that explores the stress of handing in one’s PhD Thesis. Being sponsored by Activision Blizzard, and with most of us being gamers ourselves, the decision to create a game for our final project was an easy one. Furthermore, as some of our group members are students themselves, with one in the process of handing in their own PhD thesis, this theme felt particularly fitting.



Top 2 images: Game inspiration. Bottom 2 Images: Our project Thesis Quest.

We knew we wanted to create a light-hearted and fun take on student life that brought together our love of RPGs and retro gaming, but in a very current setting. We ditched the dungeon for the college campus, and decided to use the different buildings as their own mini challenges, from a maze to a quiz game, all the while tying it to the college theme, as we’ll expand on in the “Background” section of the report.

Even though most of us had no previous experience with Pygame, we were excited to sink our teeth in, explore this library, and make it another tool under our belt. In the following sections we’ll explain how we took our first steps in game design and our idea took shape.

BACKGROUND

We conceived the game as a top-down college simulator during some initial brainstorming sessions, which challenges players to manage time and stress while navigating our custom campus map. Through a combination of RPG, puzzle, and interactive elements, Thesis Quest transforms the academic journey into an engaging if a little stressful gameplay experience.

The goal of the game is handing in the main character's PhD thesis and overcoming the challenges that come with that. After considering several possible games the idea was born to utilise each building as their own mini challenge, which not only made for diverse gameplay, but also allowed us to try our hand at several types of mini-games.

The project goal became to create a campus map with four buildings, each containing a game the player need to pass to complete their thesis and win the game. These mini-games would simulate critical aspects of graduate study, including a typing challenge representing thesis writing, a multiple-choice quiz testing academic knowledge, a maze to navigate find your way out of the IT department, and classic food fight in the cafeteria. Furthermore, there would be a wellbeing room for the player to practice mindfulness to alleviate stress.

We wanted to add a countdown timer which when runs out the game is lost. We also wanted to implement a stress metre to represent how stressful the period of handing in one's thesis can be. Subsequently, when the stress metre would max out, the game would also be lost. One way to alleviate stress would be to go to the wellbeing room as mentioned above. We decided the game could be won if all mini-games were won, within the allotted time and moderate stress.

At the end of the game, we thought the player could graduate with a Pass, a Merit or a Distinction, which a little nod to our own CFG journey, and representative of exam grading in general.

Built using Pygame, Thesis Quest is ultimately about creating a fun, interactive experience that puts players in the shoes of a postgraduate student - with plenty of unexpected twists and challenges along the way.

SPECIFICATIONS AND DESIGN

The game was broken down into 5 main parts: menus, map, character sprite, mini-games and database. These were the building blocks of our project. The game opens on the Main Menu that lets players start a New Game, view High Scores, or simply Quit. The High Scores screen connects via Flask to the API that grabs the top 10 highest scores from the SQL database.

For our main map we uploaded each building individually so we could move them and resize them to fully customise the map. We wanted the player to be able to run the sprite around the map, and into the entrance of buildings, where it would trigger the mini-games. For that to happen, collision detection and hitboxes would have to play a big part in the design, as we'll explain in Implementation.

Another important aspect we wanted to implement were trackers. A time tracker to countdown until the thesis deadline, a challenges tracker to indicate how many mini-games were passed and a stress tracker to keep track of the player's stress. We wanted the stress tracker to rise with every failed mini-game attempt and make the player lose the game if they reached maximum stress levels. To give our player a fighting chance, we wanted to include a mechanic that would allow them to lower stress at the cost of time. That's was the goal of the Mindfulness Memory Game, which makes for a welcome break from the stress of academic life.

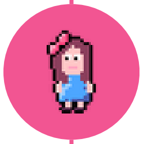
Being a game that contained 5 other games in it, we knew this project would require a lot of integration, as each mini-game has its own specifications, requirements and would still need to interact with the main map and trackers

The Library Quiz needs at least 3 correct answers out of 5 for a win. The Maze needed to be able to generate a different map every time and place random choices along the way that affected time. For the food fight we needed to create an attack/defence combat system. The typing game needed to take player input to type up the sentences. Finally, the Mindfulness game had to be able to identify collisions between the player mouse click and the coordinates of the cards. Furthermore, it needed to reveal and hide the cards as the player clicked on them, leaving them revealed if they were a match, making it quite challenging.

The character sprite was key piece in our game that would require its own class to contain its movement and animation methods, as well as the collision logic mentioned above. The Hero sprite art was designed by Gwen for the game, which she later upgraded to a sprite sheet. The asset collection, from images to music, played a big part in bringing the game aesthetic to life, and it proved to be a rather time consuming task.

KEY FEATURES OF OUR SYSTEM

This section breaks down where certain requirements have been met.



- **Start Menu/High Scores - API and Data Base**
  - Option to start new game, view high scores or quit
  - Connect to API via Flask to retrieve scores from Database

- **Libraries**
  - Pygame, Requests, mySQLconnetor,

- **Object oriented programming**
  - The majority of the code was written in classes, some main examples would be:
    - Mini-games
    - Character Class - main/character_class.py
    - Game Class - main/game_class.py
    - Map Screen - world/game_screen_classes.py
  - **Decorator**
    - Used @abstractmethod and @staticmethod in several of our classes for example the Bubble - utilities/speech_bubble_map.py

- **Interactive Campus Map**
  - **Tile Map - Itertools**
    - **Itertools.produc**t used to iterate through the tile map and draw the campus, instead of enumerate. - world/game_screen_classes.py.
  - Each building features a different mini games
    - **Library Building:** Multiple choice coding quiz
    - **IT Building**: Random Maze with choices
      - **Recursive function** - carve_path() in main/maze.py
    - **Cafeteria Building**: Food-fight combat
    - **Mindfulness Building:** Matching positive affirmations
    - **Classroom Building**: Fast Typing Game

- **Main Run file**
  - The game can be run from the file main.py which instantiates the game class - main/main.py

- **API/Database**
  - Prompts player for name at the end to save their score
  - GET Request to display High Scores
  - POST Request to save scores at the end of a winning game

- **Testing**
  - Created 5 test suites for functions that perform important tasks

IMPLEMENTATION AND EXECUTION

Our development approach began with the implementation of the software platform Jira. This allowed us to easily assign tasks whilst keeping track of the progress. We knew from the get go that managing our time well was a must as having so many moving parts would take a long time to integrate. Using agile methodology we created a sprint for the project submission, which we broke down into Epics for Main Features, Extra Features, Testing and Documentation.

We gave ourselves three deadlines: one week for idea consolidation and task division. Then, one week for first version, two weeks for refactoring and final version, with a final week for integration, testing and documentation. In retrospect, integration could have used more time. We also scheduled weekly, and sometimes bi-weekly team check-ins on Slack to track progress and reassess. Some documents like the Individual Logs, and documentation were uploaded to a shared google drive, so everyone could work on them remotely and simultaneously if needed.

Once the majority of the game's subtasks had been successfully completed, the entire group met for an extended mob-programming session. The session lasted several hours and involved live programming through screen-sharing, in-depth discussions, and rotating roles. Integration would go on to last another full week and members taking turns to avoid merge conflicts.

We then divided the initial tasks into character class and movement, world map, menus, the five minigames and eventually testing. To work simultaneously and collaboratively on these tasks we used the version control system git and created a shared group repository on the Github platform. Before we started we created necessary directories to structure the project, to keep it as organized as possible from the beginning.

To avoid initial merge conflicts, the group members contributed individually to a prototypes directory which contained subfolders labelled with their respective names. Every group member developed their features on separate branches locally, consistently committing and pushing their changes to the shared GitHub repository. Once a feature was finalised the files were moved to their destination directories.

In terms of libraries, we quickly decided we wanted to use Pycharm for our game, instead of other libraries like Tkinter. Integration was by far our biggest challenge. Having a menu, map, several games and victory and game over screens, we needed to create a game engine that would allow us to transition from one to the next and then back to map when triggered.

The first step was to define hitboxes, which are a set of x and y coordinates, plus dimensions in pixels, like 64 x 64, that define an area on the map. We created several dictionaries containing hitboxes for every building, as well as a separate dictionary that contained the building entrance hitboxes. This is when the collision logic would come into play.

We added a function that would check when two sets of coordinates intersected, namely the hero sprite, and the locations of buildings. This function would then be called with the movement handler of the character class and retrieve from the dictionary the name of the building the sprite had collided with.

We then would pass this name to the game's state engine which contains if statements that draw the map that corresponds to the passed name argument. We initially struggled with the global game_state variable, which stores this name being constantly overwritten in the main loop. To resolve this, we gave each game file their own variable initialised in the class constructor that would then be checked against the game state of the game class, and update if different. This resolved our overwriting issue, and allowed us to transition between menus, maps and games.

Next, we had to create functions to check the challenges won and stress and time meters, and call them in the loop to either trigger Victory or Game Over. One of the solutions was creating a dictionary to keep track of which games had been won, which would be checked against each games victory status variable. We also had to make sure the timer would continue to update correctly when out of the main map. This meant using the time module to track the elapsed time during a mini game and then subtract that from the timer on the map screen.

A final task was integrating the database. After setting up the API/flask, we had to make sure the POST method was saving the scores correctly. There were a few issues with the format of namely the player time. To fix this we created a function in the timer class that would calculate the time the player used and formatted it as string to pass into the database which took a VARCHAR for that column. We felt that saving high scores added to the replayability, as the gamer might wish to beat their own high score!

TEAM APPROACH TO TESTING

In terms of testing, being a game with many graphic elements, manual testing played a big role. To guarantee the functionality of the visual aspect such as the map and all menu drawings, as well as player movement and animation we had to manually test all of it, by repeatedly running the game. Regression testing was key, as we always re-ran the game after integrating a new feature to make sure all other elements are interacting correctly with, and nothing has stopped working.

This allowed us to spot bug, and tackle as they appear, and find the root cause, for a quicker fix. We also often shares screen grabs and videos on the group slack to keep the other team members up to date on the development of the visual elements.

The API was also manually tested at several stages. First, before integration, and them, as it the requests were being triggered by the menus of the game. We also decided to do an automated test for the top10_high_scores retrieval function, and for this we mocked the database end with Unittest Mock to just test the front-end python function.

We also chose to test three other functions that played important roles. Two are collision function: check_collision is used to look of collisions with buildings and stop character movement, which is what allows the hero sprite to only move on the grass. Next, it's the enter_building, which is an integral part of the game's state engine as it's use to update the game state according to which building was hit, hence it returning the name of the building, instead of true or false like the collision function. Finally, we wanted to include a test of one of the mini-game function. The is_walkable function is what allows the hero sprite to move through the maze as it checks if the next square is a path of a wall.

A lot of our testing had to do with the sprite movement, as it's a core mechanic in the game. Having an iterative approach with regression testing also proved invaluable. On this folder you can find some examples of our manual testing: **LINK TO MANUAL TESTING FOLDER**

CONCLUSION

Working on Thesis Quest was an incredible group learning experience. None of us had any extensive knowledge of Pygame when we started, but we quickly discovered how exciting game development can be. This project really help us understand the importance of communication and collaboration in a programming environment.

Each challenge we faced, whether it was implementing game mechanics, managing assets, or debugging complex interactions, became an opportunity to grow our skills. From creating interactive mini-games to managing the game's overall structure, we pushed ourselves to learn new techniques and work together effectively.

By the end of the project, we not only created a fun and unique game about the PhD experience, but we also gained valuable skills in game development and teamwork. Pygame turned out to be a rich library to explore, and we're proud of what we accomplished together.