INF-1400
Mandatory assignment 3 - Mayhem Clone

March 12, 2021

1 Introduction

Assignment 3 consists of two parts: In part one you will implement a clone of Mayhem, which is a classic Amiga game. In part two you will answer some theoretical questions.

Part one can be completed in teams of two. The team members will then deliver a joint version of the source code and the report describing the implementation. If you prefer to work alone, that is also OK, and the requirements reflect this possibility.

Part two must be answered and handed in individually.

Part I

2 Mayhem Clone

Mayhem[1] is a two-player game. Each player controls a spaceship using the keyboard, and the goal of the game is to shoot down the other player's spaceship. Four controls are available: rotate left, rotate right, thrust (engine) on and fire. The game world has gravity, so the player must use thrust to avoid crashing into the ground. In addition the game world has obstacles that "absorb" bullets. To understand how the ship is supposed to behave, you could watch this video[2] from a similar game called Gravity Force.

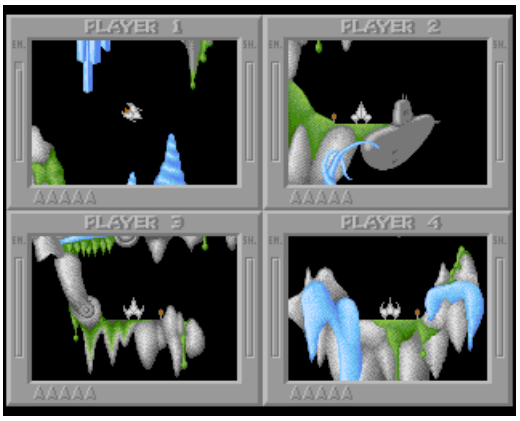

Figure 1: Screenshot from Mayhem

During research for this assignment, a most unlikely discovery was made: The author of the game is Espen Skoglund, a computer science graduate from the University of Tromsø! This is revealed in a discussion thread spanning 8 years, where a 50£ reward is promised for finding the game[3]. Espen's post can be found in the middle of page 2 of the thread.

In the early 1990s, a group of students here in Tromsø developed a network game based on the same ideas, called XPilot[4][5]. Many students spent way too much time with that game. You are free to choose the simpler graphics style similar to XPilot if you prefer that.

[1] `http://www.lemonamiga.com/games/details.php?id=2972`
[2] `http://www.youtube.com/watch?v=iXG5Bu_xmQU`
[3] `http://eab.abime.net/showthread.php?t=2128`
[4] `https://fossgames.com/xpilot-an-online-multiplayer-space-action-game/`
[5] `http://www.xpilot.org/`

2.1 Requirements

To pass this assignment, the following features must be implemented:

- Two spaceships with four controls: rotate left, rotate right, thrust, fire.
- Minimum one obstacle in the game world. This can be as simple as a single rectangle in the middle of the screen.
- Spaceship can crash with walls/obstacles/other spaceship.
- Gravity acts on spaceships (the original has no gravity acting on the bullets, but you can choose what works best).
- Each player has a score that is displayed on the screen. A player's score increases when he shoots down the opponent. A player's score decreases if he crashes.
- Each spaceship has a limited amount of fuel. To refuel, it must land on one of two landing pads. Alternatively, you can put a "fuel barrel" at a random position that is collected by the first spaceship reaching it.
- Scrolling window, as seen on the video, is **not** a requirement.

In addition, the following technical requirement must be satisfied:

- The implementation must consist of a minimum of two files. One of these shall be a *config.py* file containing global configuration constants, such as screen size, amount of gravity, amount of starting fuel.
- The main loop must have timing so that the game is playable on different computers.
- The game shall be started using Python's *if __name__ == '__main__':* idiom. Inside the if test, a single line shall instantiate the game object. All other code, except the configuration constants, shall be inside classes. This will simplify profiling and documentation generation.
- All visible objects shall subclass the `pygame.sprite.Sprite` class. The sprites shall be put into groups using `pygame.sprite.Group`. Then updating and drawing shall be performed using `Group.update` and `Group.draw`. If you are using a framework that makes this very hard, you should contact one of the TA's for an exception to this requirement. The reason for this requirement is that we want to make

sure that everyone has seen and used inheritance and polymorphism in a natural setting.

- All modules (files), classes and methods shall contain docstrings. If you are working in a team, the module docstring at the top of the file shall contain the name of both authors. The code shall contain comments to help make it readable and easy to understand as it is generally important to write code that is easy to reuse, extend or debug by others, and for this to be true the code must be well documented. To show why this is useful, you should generate html-documentation when you're done programming. This can be done using the *pydoc -w* command. This documentation is not a part of the hand in, as the T.A's can simply run this command themselves when grading your assignments.

- Design patterns: While it is not a requirement to implement design patterns in this assignment, try to look for any in your code and talk about them in your report. If you choose to include design patterns by purpose, explain the purpose of it, as well as its type and why you chose to implement it.

- The last task is to profile the code using *cProfiler*. Take a screenshot of the result and include it in the report. Give a short summary of the result and discuss where you would focus to improve the performance of the implementation.

3 Hints

- One of the first problems you must solve is how to represent and display the spaceship. The spaceship's image must represent its orientation. It is possible to draw the spaceship as a triangle, and rotate this triangle in the same fashion as you rotated triangles in INF-1100. The formula for rotating a point (x, y) clockwise through an angle θ around the origin is

$$(x, y) \mapsto (x\cos\theta - y\sin\theta, x\sin\theta + y\cos\theta)$$

- Thrust and gravity can both be represented as acceleration using vectors: thrust in the direction of the spaceship, gravity downwards. Add these two vectors to get the total acceleration. If a spaceship with initial velocity $\mathbf{v_0}$ has an acceleration $\mathbf{a}$ in a time interval Δt, then its new velocity, $\mathbf{v_1}$, is given by

$$\mathbf{v_1} = \mathbf{v_0} + \mathbf{a}\Delta t$$

In a short time interval the change in velocity is small. If a spaceship has initial position $\mathbf{p_0}$ and velocity $\mathbf{v}$, then after a small time interval the new position $\mathbf{p_1}$ can be approximated by

$$\mathbf{p_1} = \mathbf{p_0} + \mathbf{v}\Delta t$$

- Most keyboards have limitations on the number of keypresses they can register simultaneously (this number can be as low as 3). You can consider using the `pygame.key.get_pressed` method to avoid missing key events. Regardless of how you register keypresses, you should make the controls quite sensitive so the players are not encouraged to press several keys simultaneously. Alternatively, you can limit the spaceships to using only one control at a time.
- If your code runs slow, you can use cProfiler to find the bottleneck. Remember that creating new objects is expensive. Some common vector operations can be implemented without creating new objects.
- You need to figure out how the *sprite* module works. You can use it for collision detection if you like. Approximate collision detection is OK for this assignment. Simple rectangle collision is OK.

4 Report

Describe your implementation. Be precise. Describe the problems you encountered during the implementation, and how you solved them. Include class diagrams which include all methods used and describe the relation between all classes. Illustrate the class hierarchy. Is there a bottleneck? Why, and where is it?

If you work in a team, remember to include the names of both team members in the report.

5 Cheating

Remember that cheating, or attempted cheating during mandatory assignments, is considered the same as cheating during an exam. Here are some guidelines.

- Copying code is not allowed.
- Copying the design off someone or off the internet is not allowed.

- Wrong use of/missing references are not allowed.
- Getting *help* from another student to solve a problem is allowed.
- Discussing design and code problems with other students is allowed.
- Getting the *solution* (code, design, or report elements) is not allowed.

Part II

6 Questions

These questions must be answered individually. We believe it is possible to answer these questions adequately using around 600 words in total. You can use code snippets to illustrate your point.

1. What is the difference between a class and an object?
2. What is inheritance? What is the Python syntax for inheritance?
3. What is the difference between a has-a and an is-a relationship?
4. What is encapsulation? How is encapsulation handled in Python?
5. What is polymorphism? Give examples of polymorphism from the precode and the Mayhem implementation.

7 Hand in

Every student must deliver the complete assignment on Fronter in order to pass, even those who worked in teams. A student with student-id *qwe123* will put the files in a directory structure as follows:

```
inf1400-qwe123-3/
        |--src/
        |   |--all the source files here
        |   |--README
        |--docs/
        |   |--report.pdf
        |   |--answers.pdf
report.pdf
answers.pdf
```

The *inf1400-qwe123-3* folder is then compressed into a zip or tar.gz archive and uploaded to Fronter.

Deadline: 15.04.21 Before 23:59