Sheep Hearding

João Tiago Lacerda Rainho
UC: Tópicos de Programação de Jogos

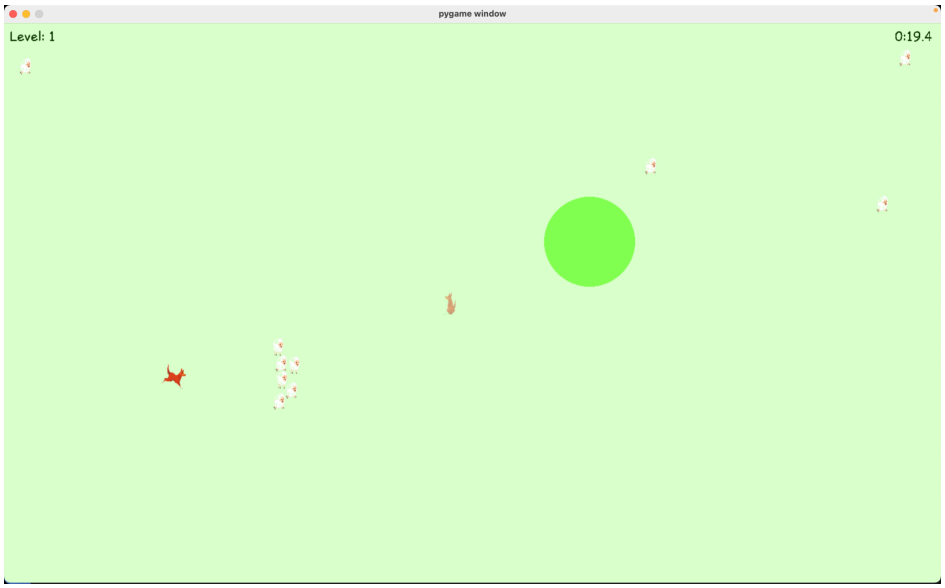
Index

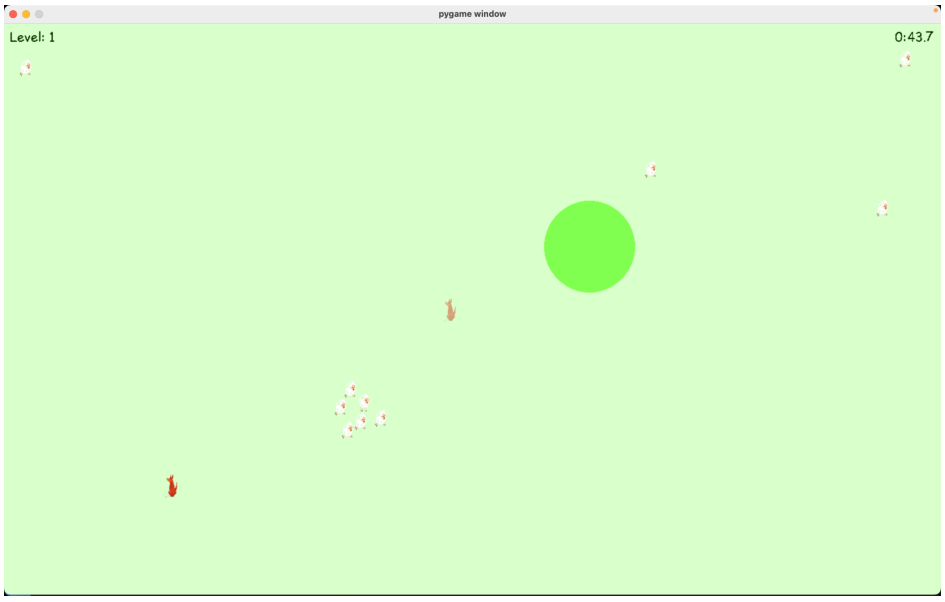
I- Game Explanation

Sheep Healing is a digital uniplayer game that takes you into the country environment to help a shepherd to control his animals. For this, the player will be able to control one or more shepherd dogs, which will scare the sheep with the objective of guiding them to their pen.

The main objective will be to complete the mission in the least possible time. Also, when all the sheep enter the corral, the player levels up, and the corral space gets smaller and smaller and the number of sheep increases, as well as the number of available dogs to control.

How to play: use “a”, “s”, “w”, “d” to control which dog to control and they arrow keys to define where the selected dog should move.



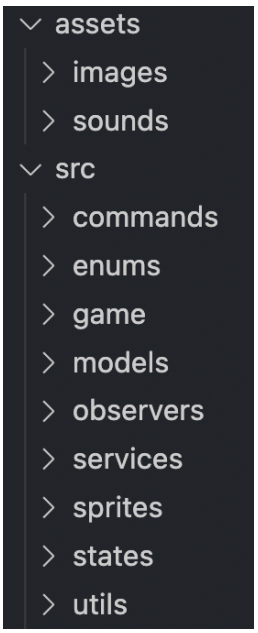


II- Code Repository

The Code of this project is available at https://github.com/tiagorainho/SheepHerding

III- Game Architecture

For a better organization, the game structure is divided into several folders (Figure 2). The two main folders are Assets, which contains the static art files; and Src, with the code logic.



In the Assets' folder there are already some important assets, such as:

- Images → Folders of each model. Inside this, there are other folders with its name as the action taken. The sprites are in order;
- Sounds → .Wav files to reproduce while playing.

The Src folder is divided into new folders for specific purposes:

- Game → Very basic game engine;
- Utils → Auxiliary classes used broadly in the game, such as the Vector class;
- Commands → Process inputs from user;
- Enums → Enumerators;
- Models → Behaviour of a certain entity;
- Observers → Event handlers
- Services → Orchestration of models
- Sprites → Define the visuals of models
- States → Finite State Machines

The main file is the src/main.py, this simply initiates a SheepGame instance and calls the start method. There is also performance debugging capabilities built in which writes the results to a log.txt file, however the DEBUG variable should be set to true.

The SheepGame contains all the service registry, using a service locator as a singleton, that the game requires in the constructor and also implements the specific game loop in its update method. This class extends from Game which provides a simplified and generic game loop.

IV- Patterns

The following table contains information about the patterns used. The “Where” column refers to the file in which that pattern was created or used.

The examples provided are only one occurrence with a brief explanation, it does not mean that it is the only usage of that pattern.

Pattern	**Where**	**Why**
Service Locator	services/service_locator	In order to access any of the registered services anywhere in the project, the services are registered in the service locator and retrieved by the service name.
Singleton	services/service_locator	Used to provide a global point of access to the service locator, the main purpose is to ensure that every time the service locator is used, it's using the same object.
Command	commands/input_handler	Abstract the complexity of connecting a key press into a command, in this case the selected dog will update based on the input of the keys “a”, “w”, “s” and “d” corresponding to the directions to search relative to the currently selected dog.
Flyweight	models/dog	All dogs receive the same DogModel object. This allows for a very low memory consumption for the amount that would be required if it needed to be used for every object.
Observer	observers/achievements	Abstract the notifications and behaviour of achievements. In this case, every time a sheep is inside a corral more than x time, the achievement is notified, and some action is taken.
State	states/sheep_fsm	It's useful to provide an abstraction layer over the lifecycle of an object. In this specific example, the sheep can either Graze or Herd depending on the proximity of threats (dogs).
Double Buffer	game/game	Only flip in memory image to screen

		once every frame. In every sprite, the blit is used to update the in memory image. Performance is also improved by only updating the current sprite image if required, otherwise the previous remains without any blit.
Update Method	game/game	The SheepGame only needs to extend Game and implement the update method and the Game will contain the basic “Game Engine” implementation, therefore the SheepGame should only contain its implementation.
Game Loop	game/game	Receives the user inputs and processes them in the SheepGame to update the game, then the Game renders to the screen.
Bytecode	sprites/sprites_model	A designer can create sprites and add in this folder, then they will be automatically used to create the animation by the filename lexical order.
Subclass Sandbox	game/game	Used for other classes that extend Game are forced to implement the update method.
Type-Object	models/dog	The dog breed is specified when creating the dog, that allows an easy change of parameters while maintaining all the same functionality of the dog and abstracting those parameters into a class.
Component	models/map_constraints	Used in map_service to constraint the objects in the game within a predefined area. It's particularly useful to decouple logic from the class it's being used on. In this case, the map service only needs to know that the map_constraints.update method constrains the objects, not how it does, and it can be used on multiple places, in this example this was only used once because there is only one map.

V- Future Improvements

If given the opportunity to improve or progress with the development of this game, I would proceed with modifying the following topics:

- Visual of the game more attractive and interactive, involving not only a general environment more in keeping with the theme, such as adapting the characters to movement.
- Improve boids algorithm, currently the algorithm is not optimized to work with a large amount of sheep. To improve that, the way to check which boids are near needs to be optimized, for example with a spacial partitioned data structure.
- Addition of new characters at higher levels, such as:
    - Wolves, which from a pre-defined time, would steal sheep if they were not kept away from the periphery.
    - Rams, whose difficulty of control by dogs is far superior because if any threat is close, they attack. Therefore, to successfully guide them, the dogs need to use sheep to lead them to the corral, otherwise they would attack the sheppard dogs, this would be an optimal exercise to use collisions;
- Multiplayer mode: have 2 corrals, on each side of the screen and the players on each team have to coordinate to bring and steal sheep to their corral in order to achieve the maximum amount of points.
- The dogs inside assets/dog_old_sprites are dogs created by me, however these sprites were not used because there were problems using those sprites with few pixels because the resolution of the sprites were too high. However, they can be used by simply loading the correct folder while creating the DogModel in SheepGame.add_level method.

VI- References

The following links correspond to the sources/inspirations for the development of the project:

Sources:

- https://wiki.spectralcoding.com/info:league_of_legends_sounds:league_of_legends_sounds
- https://wiki.spectralcoding.com/info:lol_sound_pages:summoner
- https://thumbs.dreamstime.com/b/red-dog-symbol-year-set-poses-animation-dog-dog-running-red-dog-symbol-year-vector-104276052.jpg
- https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS6QTAp7jGLXY3F-gYo54TYrT9dI522eO0N8cCpxUssQV31rpxBcCXyLq4bMTpujtFcCu4&usqp=CAU
- https://i.pinimg.com/736x/21/44/af/2144af71aa60bee0c91931bb8bafd16f.jpg

Inspiration:

- https://www.youtube.com/watch?v=xgxEDLQs7B8
- https://www.youtube.com/watch?v=bqtqltqcQhw