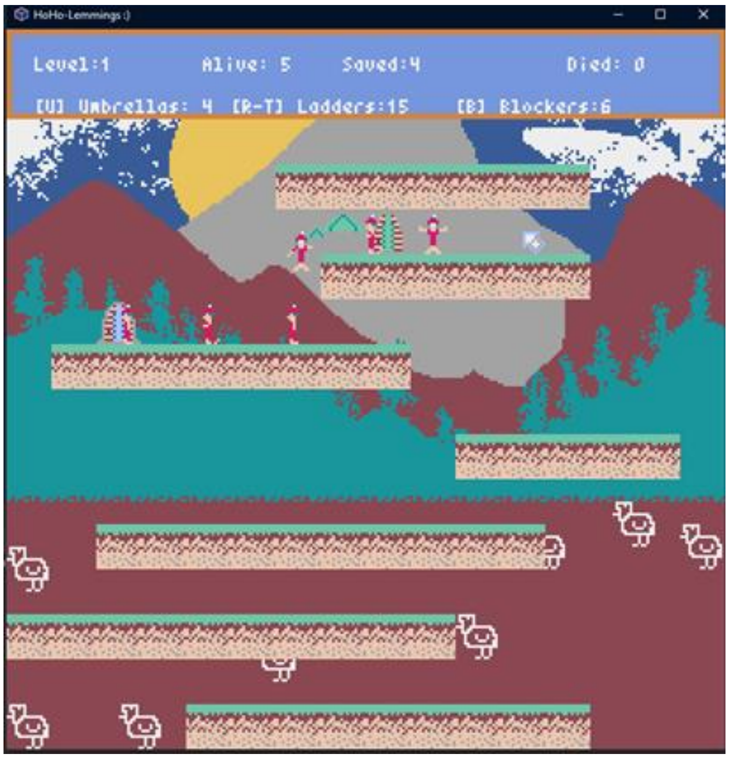


December 2020

Students:

Isaac Buiza González (Group 89)

100451176

Ruth Navarro Carrasco (Group 88)

100451032

Lemmings

Programming

Project Report

This project simulates the popular retro game *Lemmings*, published in 1991. The main goal of the game is to try the maximum number of lemmings to reach the exit gate.

Table of contents

1. Classes design

The video game ***Lemmings*** is made up of 9 classes, which have been thought in order to achieve a better organization and understanding of the project.

The main class of the video game is the ***Game*** class, which holds all the logic of the game. This class contains three methods: a constructor or init method, where platforms, gates and the grid will be created; an update method, which will be in charge of upgrading the videogame in each frame; and a draw method, which will be in charge of drawing all the map of the game.

Another class of the game is the ***Lemming*** class, which contains each of the attributes a lemming object has. For instance, a lemming has a position (an “x” and a “y” coordinate) and a direction. Once a lemming object is created this attributes must be given. There are also other attributes that are given by-default like their life status, the tools a lemming might have, if it´s falling, etc.

A class ***Cell*** has been implemented in the game in order to create the grid. This class has an attribute for its position, another attribute for each type of tool (umbrella, blocker, left ladder, right ladder) and the platforms. The attributes for the tools and for the platforms are given by-default values which are False.

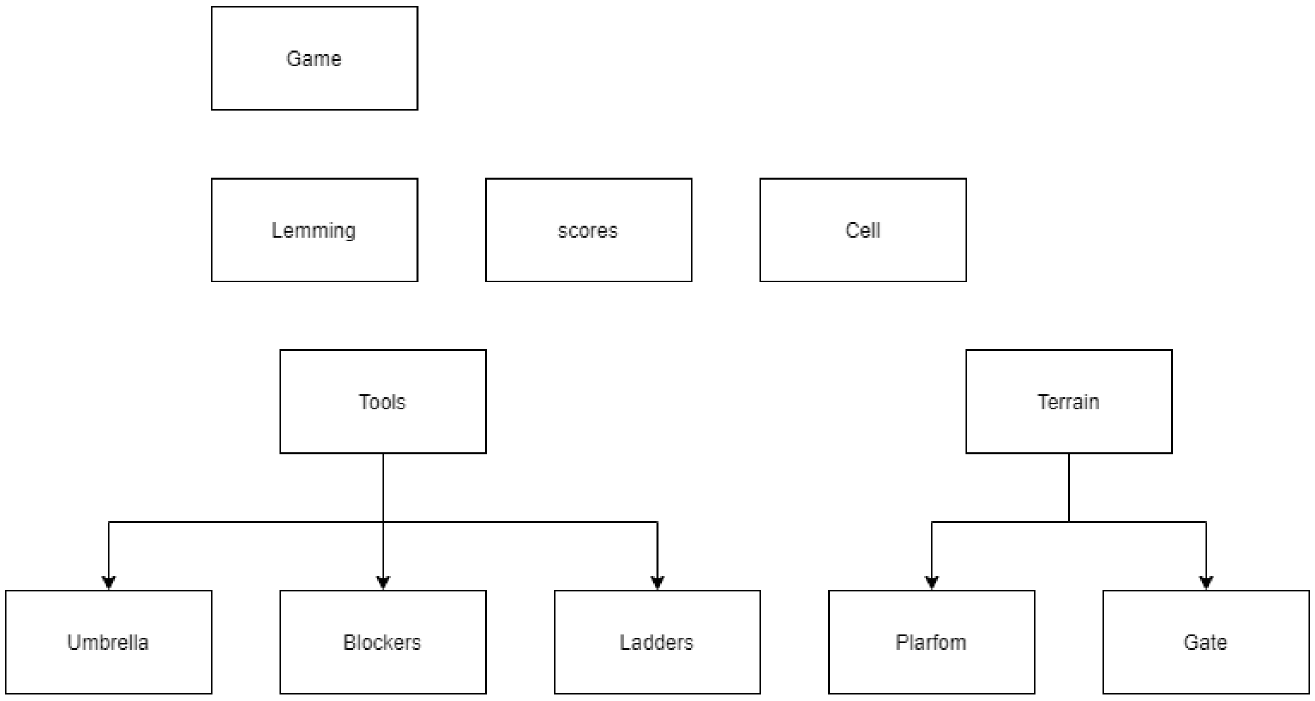
A class ***Scores*** has been established in order to update the records. The attributes of this class are the level, how many lemmings alive are, how many lemmings are dead and how many lemmings have been saved.

Also, a class for the ***Platforms*** has been created. This class contains attributes for the position where the platform is going to be generated, which will be pseudorandom, and also the width, which will be also random. It also contains an image attribute.

Furthermore, a class for the ***Gates*** has been implemented. This class holds attributes for the coordinates and the type of gates that are going to be generated (a start or exit gate). It also contains an image attribute.

Three classes have been created for the functionality of the tools. The classes ***Blocker*** and ***Umbrella*** contains the same attributes although they perform different functions. They have attributes for the coordinates and for their active state. They also have an attribute, which is a tuple, for the images. The class ***Ladder*** has the same attributes as the Blocker and Umbrella but, as it is a more complex class, it has another attribute which is construct, which is given a by-default value False.

Figure 1. Class structure



2. Most relevant methods

In order to achieve a good functionality of the game the following methods have been created.

As a common method; all the items (tools and lemmings) have an **imageUpdater** that allows each object to have a specific animation regarding its status; for example, if a lemming is falling with or without umbrella it will have a different animation.

Inside the ***Scores*** class we have created three methods, one for each tool, that return the number of usages available for each tool. In order to compute this number, the maximum number of the tool and how many of them have been placed are taken into account.

The class ***Ladder*** has a **constructLadder** method that updates the building status of the ladder each time it is executed and returns True when the constructing process is finished.

The class ***Cell*** has a **checkGrid** method that returns if a cell is free or if it has a tool and the kind of tool it has.

We have created methods related to the movement of the lemmings. They can be found inside the ***Lemming*** class.

- The logic inside the movement of the lemmings is the following: the lemmings are constantly moving along the x axis if they are not falling, no matter the direction they have but, if they are in a ladder, they will be moving in a diagonal path, meaning that they will be moving in both x and y axis.; if they are falling they will move along the positive direction of the y axis. This method has been named as **movement**.
- The **notfalling** method allows to change the attribute of the lemming’s life status if they fall to a platform, depending whether the lemming has an umbrella or not.

The following methods can be found inside the ***Game*** class.

- The **init** method is where the grid, the platforms and the gates are created. Also, in this method, we create the coordinates for the cursor in order for the player to move around the grid.
- The **exist_floor** method allows the program to detect if the lemming is above a platform or not. If the lemming is not above a platform, it will start falling and invokes the "**notfalling**" method.
- The **tool** method is in charge of invoking a method for each type of tool.
    - **blockerUsage** method is used for activating a blocker object. The first lemming that arrives at it will be transformed into a blocker, by moving its object to a list (lemmings_blockers); the following ones will change their direction when they collide with it.
    - **umbrellaUsage** method is used for activating an umbrella object. The first lemming that reaches the umbrella activates it. All the lemmings that go through it will receive an umbrella that prevents them from dying in the next fall.
    - **ladderUsage**. In our main code, this is divided in two methods, one for each type of ladder, but they work in the same way. The first lemming that arrives will start the constructing of the ladder and will ignore it. Once the ladder has been constructed the lemmings will be able to go up or down it.
- The **check_tool** method allows the program to place or replace an existing tool in a cell into another one. If a tool has been activated and it is a ladder, nothing can be done, but if it is a blocker or an umbrella it can be replaced, but the scores will have been reduced.
- The **update** method is updated in each frame. This method carries out the main logic of the game, taking into account keyboard interactions and the interaction of lemmings with the environment, such as platforms and tools.
- The **draw** method draws the map where all the game will take place, the updating scores and also will be in charge of drawing the animations (invoking each object`s "**imageUpdater**" method) of the tools and the lemmings.

3. Most relevant algorithms

Taking into account the way pyxel implements the update and draw methods, we have designed them in the following way. The update method is divided into two sections: the first part is in charge of the inputs of the user and the actions these performs, like moving the cursor or placing tools; the second one is in charge of the lemmings' interaction with the environment. The draw method carries out one of the most visual function of the program, as it is in charge of drawing every object that has been created inside the class ***Game***. Also, the cursor is the last thing that is drawn in order to not lose it.

All the tools have in common the same activating algorithm. As soon as a lemming reaches a cell where a tool has been placed, it will be activated and allow them to use them.

Another feature is that when the lemmings reach a ladder, they have to follow a diagonal path. We have implemented this by adding a vertical increment to the normal x axis movement that the lemming has.

In addition, to solve the problem of the user placing tools in the same cell as the platform, we implemented into the tool placing method (**check_tool**) a condition that checks the platform attribute of the target cell and prevents the user from placing a tool if the cell is occupied by a platform.

4. Work performed

We have been able to achieve a good functionality and implementation of the game and we have reached all the required steps of the game. We tried to build our code in the simplest possible way in order to be more understandable.

4.1. Functionality

The goal of the game is to allow the maximum number of lemmings to reach the exit gate without dying, which will occur if they fall from any height, including user-built ladders. To achieve this the user/player can place tools that have a defined usage.

In order to start the game, the user has to press the letter S for the lemmings to start going out the start gate with their respective animations. After starting the program, the user can start playing by placing the tools in the cells that founds more suitable in order for the lemmings to reach the exit gate. The keys for placing the tools are the following: "U" for umbrella, "B" for blocker, "R" for right ladder, "T" for left ladder. The user can place tools before the lemming's spawn, in order to prepare the level to be played.

We have also created different sprites in the tools in order to know when they are activated or not. The closed umbrella means that it is not activated, but once a lemming passes through the cell that contains the umbrella, this image changes into one with an open umbrella, meaning that it has been activated. The stop signal is the blocker and once a lemming passes through the cell, it will be transformed into a blocker. The white ladder means that it has been placed and once a lemming activates it, it will start constructing it.

The lemming has implemented animations for each action. For instance, if the lemming falls from a platform without any umbrella, it starts making loops. Also, if it falls from a platform with an umbrella it has an animation opening the legs. They also have an animation for walking, in which the lemming moves its legs and its arms.

We have also made animations in the ladders in order to construct them. Until the first ladder of the chain is not completely construct, the lemmings cannot start using it, but with the following ones they can stand over the steps that are already built.

If more than 10 lemmings reach the exit gate, the level is passed, otherwise the player will have lost. Also, the game is also over if all the lemmings die.

4.2 Parts not performed or functionalities provided

We have implemented in our game an auto- destruction button that is original from the game. By pressing the key "K" all the lemmings will die.

We have totally implemented the requirements of Sprints 1 to 4. We have not added any additional functionalities in our game from Sprint 5.

5. Conclusions

5.1. Main problems

We found a lot of difficulty in the tools, specially in the ladders. After achieving the diagonal movement of the lemmings, they didn't detect the end of the ladder, so they continue with this diagonal movement in an infinite way. We fixed this error, but we found another, the order of the construction of the ladders matter. But we never give up and we develop the whole cell class in order to solve this problem.

We notice in an advanced status of the project that we have not implemented any getters and setters and we realized, luckily, that this was wrong.

We also noticed that we had generated in the wrong way the matrix for the grid. We noticed that had the columns and rows inverted, but we were able to fix it on time.

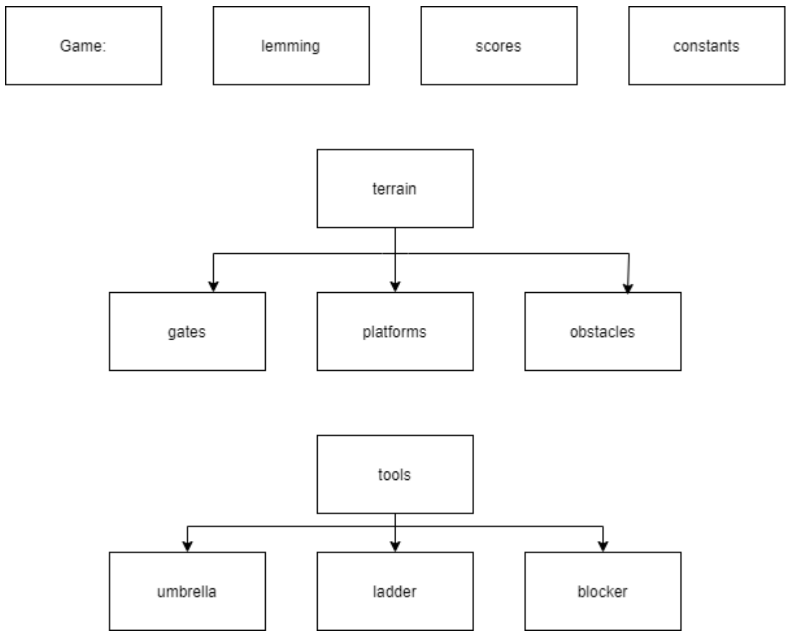
5.2. Personal comments

This taught us the basics of OOP. We are aware that a lot of attributes and methods are shared among similar classes; this could be improved implementing inheritance. We could not have done this because we did not have the knowledge at the right time.

Some bugs are present in the behavior of the ladders. For instance, when two stairs are placed forming a V shape one of them is ignored by the lemmings; this might be due to the stairs occupying a little bit more than a cell, but we were not able to develop a solution to this problem.

Other function that could have been implemented is the possibility of making the blocker lemmings move again like in the original game. This can be done by relocating the lemmings object from the blockers list into the alive ones again.

We have not followed our original structure of classes designed as we did not have the necessary knowledge and ended up splitting all the classes that required inheritance into different ones. The original structure of the program was the one as the following attached figure.


Game:
lemming
scores
constants
terrain
gates
platforms
obstacles
tools
umbrella
ladder
blocker