2805ICT PACMAN MILESTONE 3

Sebastian perry

S5132483

At a glance

This document will serve as a progress report on the steps taken to implement the Pacman game from the perspective of being at the end of the development period

The video is posted at: https://youtu.be/0jM8jMj51bE

Contents

Introduction

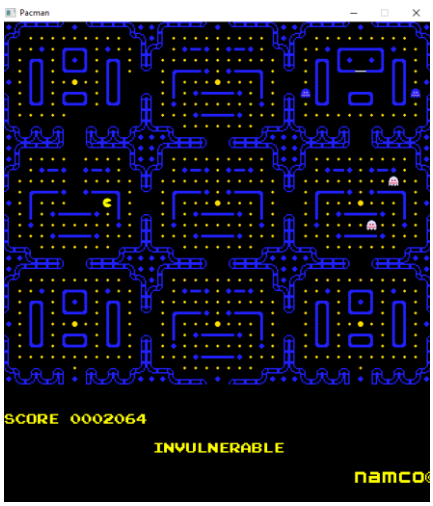
All of the following implementations of the Pacman game have been done in c++ with the use of a family of packages which live collectively under the name of SDL.

Implementation of Task 1

For task 1 Pacman was implemented in a grid fashion as requested producing the game which can be seen in the screenshot below, and at time stamp 0:00 in the video (please bear in mind that these medias were generated after the implementation of task 2, and 3; as such, features like the random maze generation, and the score decline are still present).

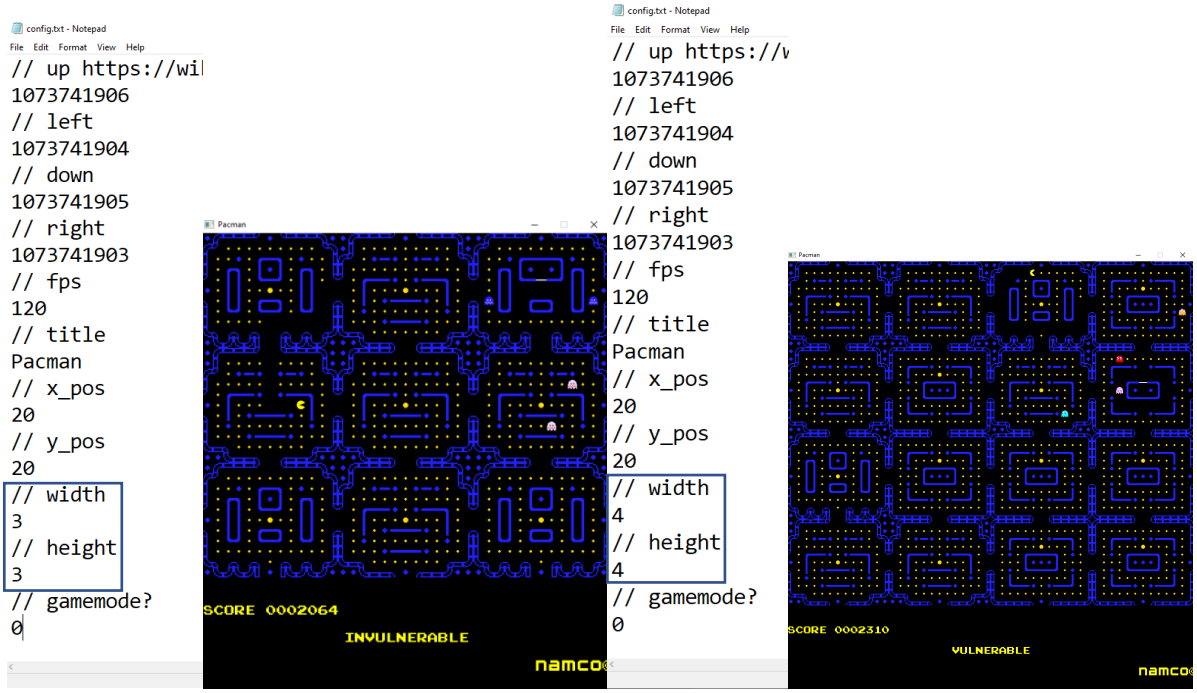
Implementation of Task 2

For task 2 sadly the Implementation of the different view types remain unfinished with the hexagon view remaining 100% unimplemented, and the graph version in a state which would work if not for an unfortunate bug relating to its collision detection of walls (footage of this can be seen at 4:10). With this being said however, it can be seen in the picture below, and more importantly the video at the time stamp 3:40 that a score system was added which awarded different decreasing amounts (as the time went on) of points for the players eating the various other entities (please bear in mind that these medias were generated after the implementation of task 3; as such, features like the random maze generation are still present).

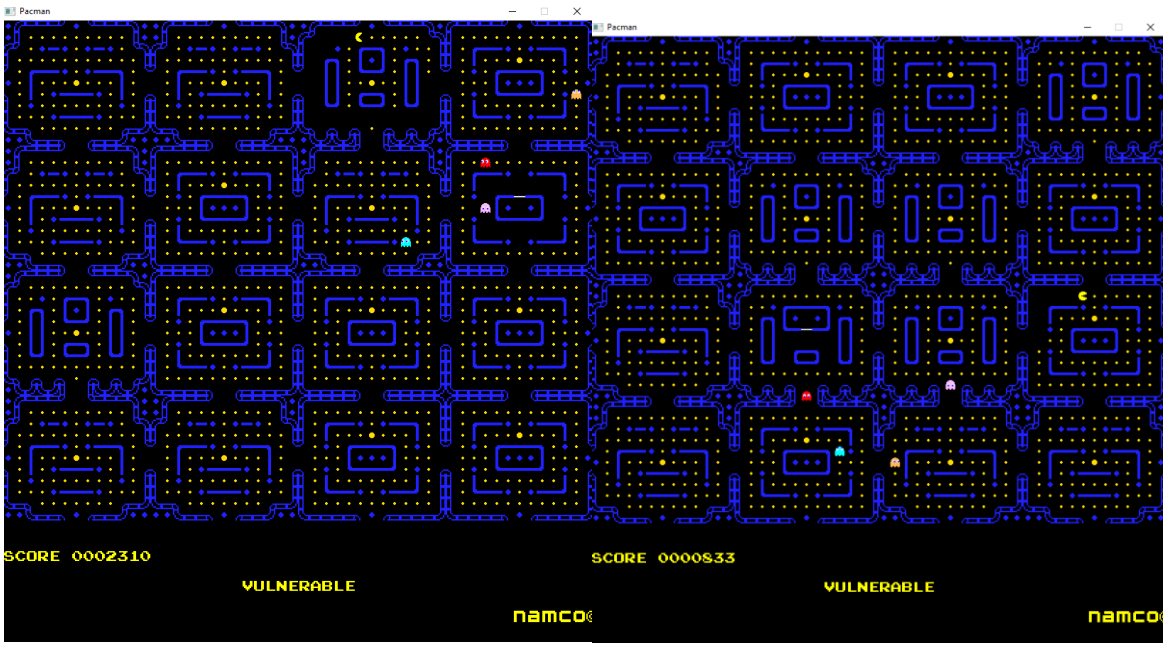


Implementation of Task 3

For task 3 the game was altered so that a number of different features could be added. First of all it was possible to edit some of the game options through altering the contents of the config.txt file (found in the rescource_files folder) (3:55 in the video):



Second the game was edited so that the map would generate random combinations of the map segments insuring that the game is always different on successive playthroughs (3:55 in the video):



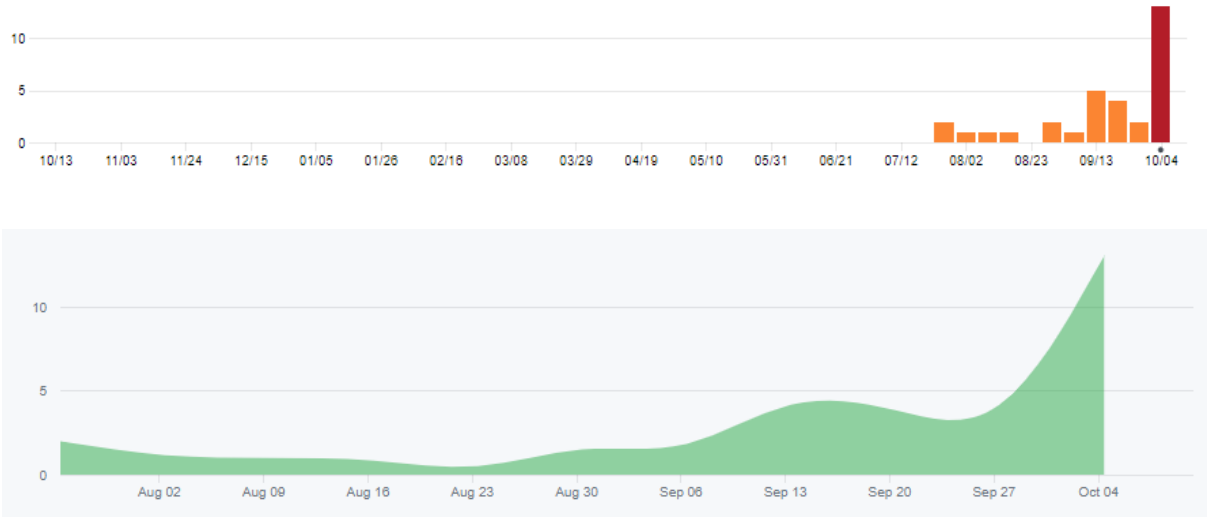
Lastly more sophisticated ai was attempted with the an implementation of A* being given to ‘blinky’ (the red ghost). Sadly however, upon examining the gameplay it seems as if the implementation wasn’t fully successful in practice.

Cross platform

As of the submission of this project, cross platform compatibility was no possible. With this being said, as this game was created using 32bit packages, it should be able to run on both 32 bit and 64 bit versions of windows

Software Version control

Over the final implementation of task 3 the previously used version control (git through the server of github) was continued to be used. The below graphs show the use of this tool over the development period:



As of the submission of this project, no system of version control had been implemented.

Objective 1

In the project many design principles were used to aid in the implementation of the project. First of all c++ was used for the implementation of the game which allowed for the following of the Principles of ***least privilege*** through the use of private class contents, allowing the fulfilment of the ***failsafe defaults*** principle by enabling classes to hide unnecessary data from other classes. Furthermore, in following the principle of ***separation of concerns*** (where different modelling elements were separated for clarity), it became necessary for certain classes to be available in multiple places at the same time; it is this need which lead to the implementation of the singleton model with a few of the classes found within the project.

Display of map recreations and sprite sheets

As stated before the game is currently experiencing a breaking change that is preventing it from displaying it's graphics, however the below sprite sheet and level recreations will be shown.

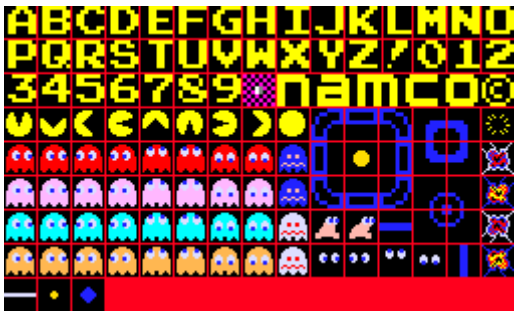


Figure 5sprite sheet

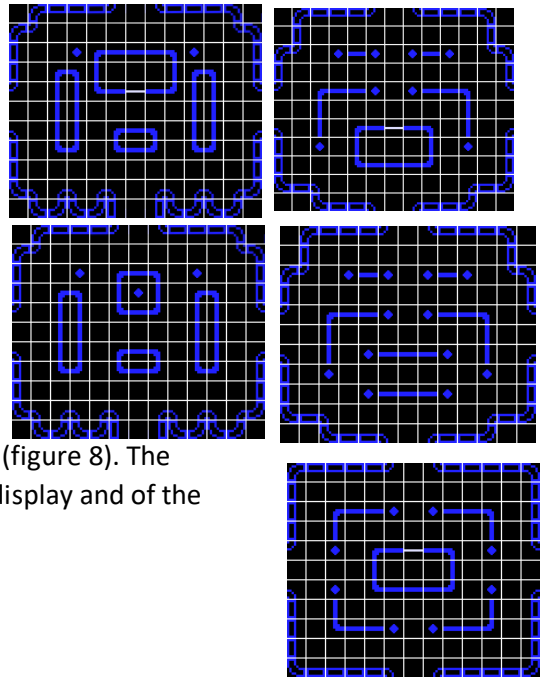


Figure 6 map recreations

In order to create the maps shown to the left sprites are defined by location and stored in an array (figure 8). The indexes of this sprite locations can be later called to display and of the sprite, or in this case entire maps(figure 7)

```
// generate the playspace's static instance indicator
PlaySpace *PlaySpace::instance = 0;

// the map segments with a ghost spawn in it [#][y][x]
int spawn_map_segments[10][11][13] = {

    // courtyard
    {{ 48, 38, 38, 38, 38, 43, 58, 46, 38, 38, 38, 38, 39 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  47, 60, 53, 54, 54, 57, 58, 57, 54, 54, 50, 60, 41 },
    {  47, 60, 55, 58, 58, 58, 58, 58, 58, 58, 55, 60, 41 },
    {  43, 60, 57, 58, 53, 54, 56, 54, 50, 58, 57, 60, 46 },
    {  58, 60, 58, 58, 55, 59, 59, 59, 55, 58, 58, 60, 58 },
    {  40, 60, 57, 58, 52, 54, 54, 54, 51, 58, 57, 60, 49 },
    {  47, 60, 55, 58, 58, 58, 58, 58, 58, 58, 55, 60, 41 },
    {  47, 60, 52, 54, 54, 57, 58, 57, 54, 54, 51, 60, 41 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  45, 44, 44, 44, 44, 40, 58, 49, 44, 44, 44, 44, 42 }},

    // skull keep
    {{ 58, 48, 38, 38, 38, 43, 58, 46, 38, 38, 38, 39, 58 },
    {  48, 43, 60, 60, 60, 60, 60, 60, 60, 60, 60, 46, 39 },
    {  47, 60, 60, 57, 53, 54, 54, 54, 50, 57, 60, 60, 41 },
    {  47, 60, 53, 55, 55, 59, 59, 59, 55, 53, 59, 60, 41 },
    {  43, 60, 55, 55, 52, 54, 56, 54, 51, 55, 55, 60, 46 },
    {  58, 60, 55, 55, 58, 58, 58, 58, 58, 55, 55, 60, 58 },
    {  40, 60, 55, 55, 58, 53, 54, 59, 58, 55, 55, 60, 49 },
    {  47, 60, 52, 51, 58, 52, 54, 51, 58, 52, 51, 60, 41 },
    {  47, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 60, 41 },
    {  45, 40, 49, 40, 49, 40, 58, 49, 40, 49, 40, 49, 42 },
    {  58, 45, 42, 45, 42, 47, 58, 41, 45, 42, 45, 42, 58 }},
```

Figure 7 maps stored as a 2d array of ints

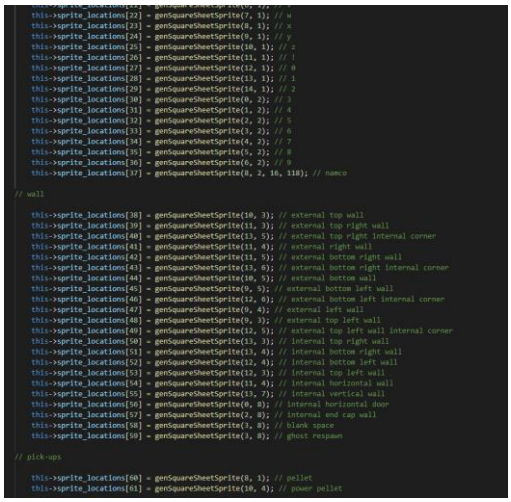


Figure 8 stored sprites