PlayDesigner

Table of contents

1.1 Abstract

PlayDesigner is an application made especially for game developers who want to make games without using prebuilt engines . For them I made Playdesinger.

This application is a valuable tool that has an intuitive and easy to understand interface that allows developers to plan and design their games .

More over the app uses a combination of keyStrokes and mouse related work to give users the most comfortable experience designing their games , with easy to remember keys and the ability to navigate the interface only with their left hand , so that the right hand is on the mouse at all times.

1.2 Background and related work

The reason I set out to build this software was due to the lack of information about how game studios designed their games before switching to engines.

What I mean by that is that there was a period before engines where people would make free form games , and today even if we search online we still don't have any clue , from all we know they could've been coded and written by hand, every single one of their levels . And I know that's not true , because games like Jack Rabbit 2 come with a level editor , made exclusive for that game, which means that other game studios made their own systems of editing levels .

This, even tho is not highlighted , is a problem because on youtube watching people make games in different programming languages they would actually write every single entity in their game , which is really messy , not to mention time consuming , because after every single change , you have to compile the game

and get to that part , check if it's alright and if it's not , you have to get through this once more .

Throughout the course of this document, you will learn about this software's research process, development plan, implementation, and conclusion.

2.1 Game design theory

Quote from Wikipedia

Game design is the process of creating and shaping the mechanics, systems, and rules of a game. Games can be created for entertainment, education, exercise, or experimental purposes. Increasingly, elements and principles of game design are also applied to other interactions, in the form of gamification.

Imagine designing a game without having any clue what it looks like, wouldn't it be funny, I'm not gonna mention that the best games have tens of levels.

Some examples of I've seen of people writing levels by hand are in these videos:

https://www.youtube.com/watch?t=220&v=PC_pAgJopIA
https://www.youtube.com/watch?t=3185&v=4q2vvZn5aoo

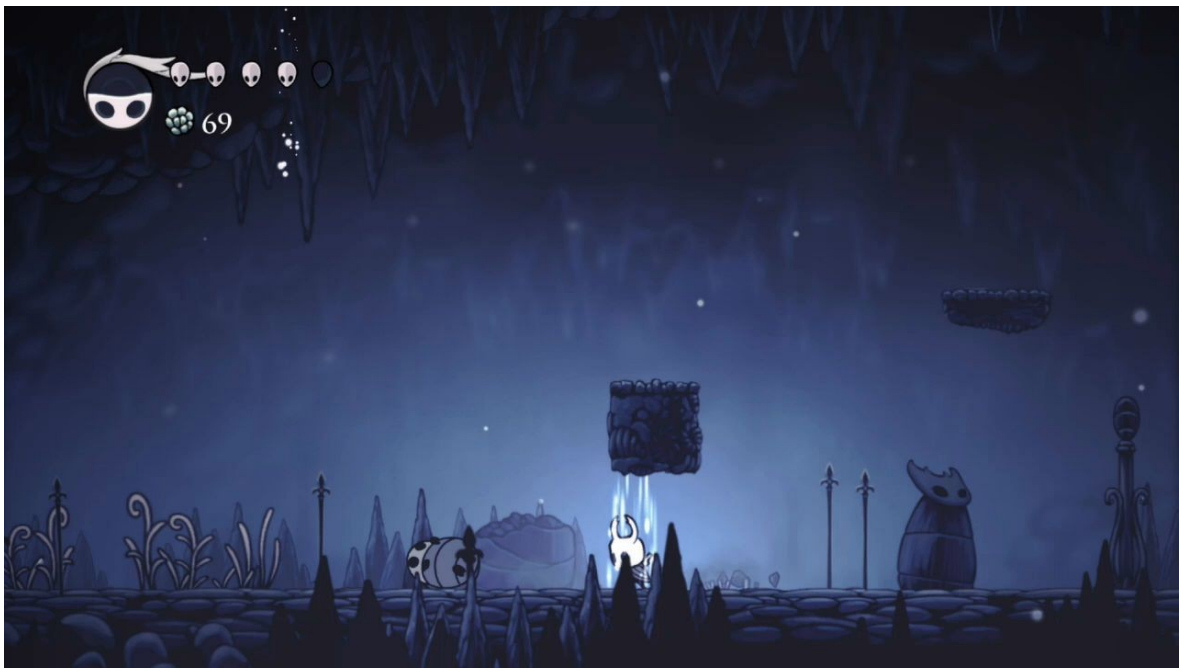
Alright these guys were making just a handful of levels for their games , and I still think it takes a lot of trial and error even for these simple games , I can't imagine someone making a metroidvania in a language like C or Python , because it's impossible , this genre is known for having a huge world to discover and different ways of finishing the game , You'd need a design tool for this , that's why I made PlayDesigner , for people who wanna make games and optimized games , after all

the tool could be used to develop games in assembly if you wanted to .

2.2 Free form games

A free form game , I'm sure everyone's familiar with is Hollow Knight , it's a metroidvania style game , developed by Team Cherry with the use of the Unity engine , and it's amazing, but then again , we couldn't recreate this kind of game without the help of an engine because it requires way to many path reviews on the story and on the map. For reference this is what the map looks like :

And this is what a level looks like :



We can break it down to :

background elements(stalagmites)

midground elements(the playable part with the knight and all of the platforms)

foreground(stalactites)

Now , I don't think you can make this kind of game without the use of an engine. Unless you are using PlayDesinger .

3.1 Background and analysis

I started by researching many ways of implementing information like position in games since the last year , on that time I was trying to create something similar to this application , but i couldn't , I picked Javascript ,and started designing , I managed to get some form of moving 3 cubes on the screen , but that was it.

One year later , here I am trying to do it again , my journey starts on May/1st/2023 with new knowledge about game development. I was willing to try again.

Since last year I learned my lesson that javascript is not the language for this kind of project, I decide it to use C++ and the raylib library to help make my application.

3.2 Development strategy

Raylib as a library has a lot of great feature , but I decide it to not use all of them , because I figured it out that I can code them myself , even though it was way harder to make them from scratch , I think I got a better understanding of how my application works.

The only functions that I used out of raylib were:

- **DrawTexture( )**
- **GetScreenWidth( )**
- **GetScreenHeight( )**
- **GetMouse( )**

I also needed to make a camera , with zoom in and out , and the ability to move that camera , raylib indeed has a Camera2d struct for that , but I managed to make a camera on my own without the use of that pre made one , and with my implementation , it actually works better .

To make the objects move with the mouse i made a class for them which has a drawing function as well as a HandleDragging function , each element having it's own variables like xPos , yPos , texture, etc.

Example:

```
3   Vector2 CameraOffset = {0,0};
4   float cameraZoom = 1;
```

Here I'm creating variables for the camera

```
10  class Object{
11      public:int xpos,ypos;
12      bool isDraggingSquare = false;
13      Vector2 offset = { 0, 0 };
14      public:Color color = RED;
15      public:Texture2D texture;
16      public:std::string SpriteName = "undefined";
17
```

This is how I declare the class of objects and its variables,very important is the SpriteName

```
18      public:void DrawObject() {
19          DrawTextureEx(texture, Vector2{(xpos - CameraOffset.x)*cameraZoom,
20          (ypos - CameraOffset.y)*cameraZoom}, 0, cameraZoom, WHITE);
21      }
```

Here you can see how I use the camera variables to draw the texture of the object , as you can see I'm just subtracting the camera offset out of the position variables and multiplying the result with the cameraZoom, which will make more sense later.

```
public:void HandleDragging(int gameScreenWidth,int gameScreenHeight) {
    Vector2 mouse = GetMousePosition();
    Vector2 virtualMouse = { 0,0 };
    virtualMouse.x = (mouse.x - GetScreenWidth() + gameScreenWidth + 25 + CameraOffset.x) ;
    virtualMouse.y = (mouse.y - 25 + CameraOffset.y);
    virtualMouse = Vector2Clamp(virtualMouse, (Vector2){CameraOffset.x, CameraOffset.y },
    (Vector2){ (float)(gameScreenWidth+CameraOffset.x), (float)(gameScreenHeight+CameraOffset.y )});
```

In this snippet , I'm declaring a virtualMouse which only works inside the canvas , and its implementation to work with the cameraOffset.

```
if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON)
&& CheckCollisionPointRec(virtualMouse,
(Rectangle{xpos*cameraZoom,ypos*cameraZoom,texture.width*cameraZoom,texture.height*cameraZoom}))) {
    isDraggingSquare = true;
    offset.x = (virtualMouse.x - xpos);
    offset.y = (virtualMouse.y - ypos);
}
if (IsMouseButtonReleased(MOUSE_LEFT_BUTTON))
    isDraggingSquare = false;
```

Here you can see how I'm handling dragging.

```
63  void Initialize(const std::string& folderPath, const std::string& extension){
64      DIR* dir;
65      struct dirent* entry;
66
67      dir = opendir(folderPath.c_str());
68
69      if (dir != nullptr) {
70          std::string FilePath;
71          while ((entry = readdir(dir)) != nullptr) {
72              std::string fileName = entry->d_name;
73              if (fileName.length() >= extension.length() &&
74                  fileName.compare(fileName.length() - extension.length(), extension.length(), extension) == 0) {
75                  FilePath = folderPath + "/" + fileName;
76                  sprites.push_back(FilePath);
77                  std::cout << FilePath << std::endl;
78              }
79          }
80          closedir(dir);
81      }
82  }
```

This function loads the png files out of the "sprites" folder .More precisely it takes their filePaths and adds them to a vector "sprites" which contains all the paths to all the png files.

Now for the camera:

```
245     static int cameraSpeed = 10;
246     if(IsKeyDown(KEY_UP)) CameraOffset.y -= cameraSpeed;
247     else if(IsKeyDown(KEY_DOWN)) CameraOffset.y += cameraSpeed;
248     if(IsKeyDown(KEY_RIGHT)) CameraOffset.x += cameraSpeed;
249     else if(IsKeyDown(KEY_LEFT)) CameraOffset.x -= cameraSpeed;
250     if(IsKeyPressed(KEY_BACKSPACE)){ CameraOffset={0}; cameraZoom = 1;}
251 
252     if((cameraZoom > 0.3 && GetMouseWheelMove() < 0) || (cameraZoom <= 3 && GetMouseWheelMove() > 0))
253         cameraZoom += ((float)GetMouseWheelMove()*0.05f);
```

This snippet controls the entire camera, you can move it using the arrow keys , and you can also zoom in and out with the mousewheel.

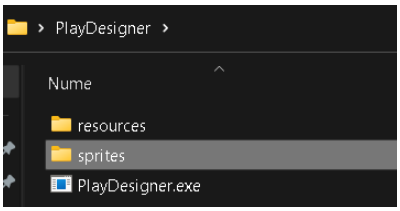
```
160 void SaveStatus(){
161     std::ofstream SaveFile("save.txt");
162     for(int i = 0; i < objects.size(); i++)
163         SaveFile << objects[i].SpriteName << " x: " <<objects[i].xpos << "  y: " << objects[i].ypos << std::endl;
164     SaveFile.close();
165     std::cout << "\n Progress Saved \n";
166 }
```

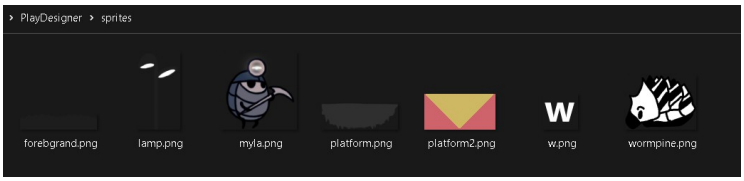
And here is the Save function , which works using the fstream library , and yes I have a Load function much longer than this , and yes expanding on them I could make a level loading system which saves and loads from a level(nr).txt and not just from/to save.txt , but in a month that wasn't a priority.

4.1 The Interface

The interface is easy to navigate , it’s designed to be used with the left hand while the mouse is in the right hand.

4.1.1 The Sidebar





To navigate the Sidebar you can use the W and S keys.

The sidebar is displaying in a list all the png files into the "sprites" folder alongside their names.

4.1.2 The commands list

This list is placed at the bottom of the screen where it's easy to observe at all times.It contains all the keys needed in using the application.

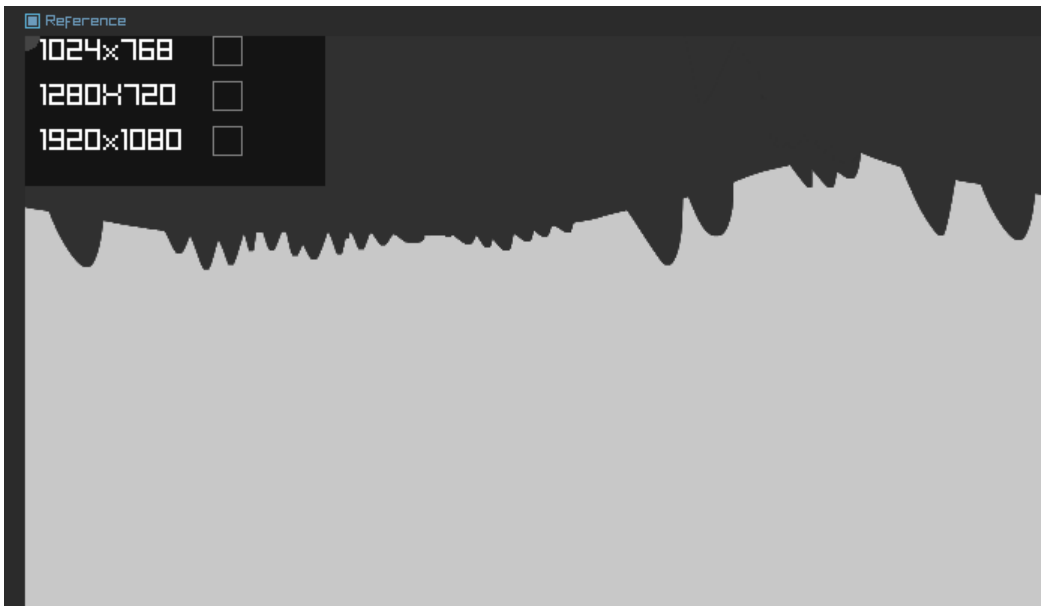
4.1.3 The canvas

The canvas is the main part of the application , it allows users to see and adjust their sprites on the screen.

It comes with some functionality , it can be zoomed in and out.

And it can export an image of the level by pressing SHIFT.

4.1.4 Reference



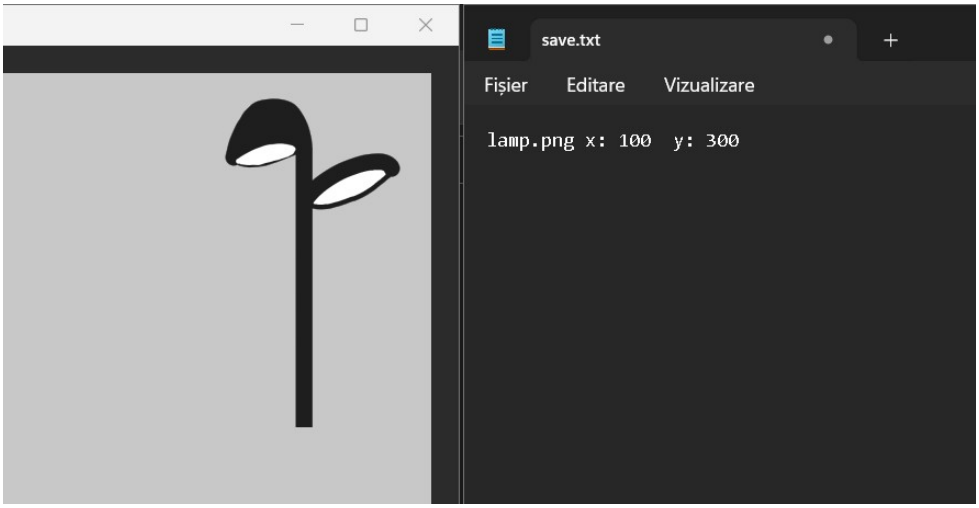
The reference checkBox , allows you to activate a resolution for reference which is gonna be under the form of a red Rectangle of the size of the selected resolution.

This is how the reference resolution looks like.

4.2.1 Save as .txt file

PlayDesigner allows users to export a .txt file with all the names of the sprites along side their coordinates , for example :

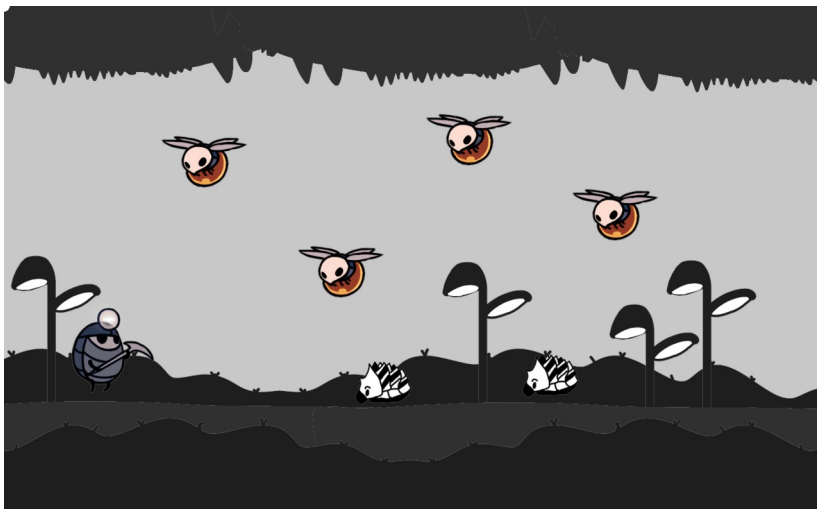
sprtie.png x: 100 y: 100



To save în this format, press ENTER
Pressing R, reloads the file and all the progress will be lost.

4.2.2 Export as image

Pressing the SHIFT key saves an image of the canvas , which allows users to capture their levels as thumbnails, being a usefull feature for analysing levels.



5.Summary

To summaries everything that has been stated, PlayDesigner is an application that allows game developers to design their games without the need of a prebuilt engine. This tool has an easy to navigate interface with lots of usefull features like:

-saving as .txt file all the coordinates of the objects, that are în the canvas

-saving an image of the level

-moving and deleting sprites

-reference resolutions

-layering elements