worksheet: Critter crossing

UFCFXG-30-2 more Games in C++

The aim of this worksheet is to introduce you to more interesting uses of the mouse in SFML games. Completing this worksheet will teach you some of the functions and logic that goes along with mouse use as well as remind you of some of the basics that were covered in your first year C++ module.

Your code will look different to mine, so there will be points where you have to figure out where to put things/how to make them work/what extra code you may need.

The first part of this worksheet relies on knowledge from Games in C++ last year, the relevant worksheets are available alongside this one on Blackboard, as you’ll likely need a refresher.

In this worksheet you will make a small “Papers Please” style game using SFML, where you have to compare passport pictures to animals and either accept or reject their entry to your country. If you’re unfamiliar with Papers Please, have a look at its website here: <https://papersplea.se/>

This worksheet is marked as part of your portfolio for this module; you can find the marking rubric in the assignment spec on blackboard.

A screenshot of a video game

Description automatically generated

# game menu and start of game loop

To start, you’ll need to create a menu screen that your player enters by default, showing the title of the game and giving options to start a game, or to quit. Feel free to call your game something other than Critter Crossing, but try to make it thematic!

Obviously there’s no game yet, but set up a background sprite that you can put your game on top of and some ENUM game states that you can put your game functionality in.

# generating animal/passport pairs

## Sprite & texture variables

To get started with the actual game, you’re going to set your character and passport sprites up as pointers in your Game.h file, like so:



Making sure that you initialise them in your game’s init() function:



We’re only going to be using one character sprite and one passport sprite for this game, changing the textures when we need to. This way we don’t need to have a bunch of sprites loaded into memory, only the textures.

Next, set up some arrays to store the animal and passport textures, I’ve supplied 3 passports so my arrays are size 3, but you can make more with the other images provided if you like.

Many more animal sprites are available in the Kenney asset packs that are in the template repository. If you add new images or other files, you might need to select “configure cache” from the project drop-down menu before they work with your game.

A black background with white text

Description automatically generated

Then, in your game.cpp, load all of the animals and passport images into those texture arrays, **making sure they’re in the same order**. If the penguin animal is in animals[0], the penguin passport needs to be in passports[0], etc. If the animal and passport textures are in different orders, the game won’t work! (This isn’t the best way to code this, but this is a worksheet, let’s keep it simple. You can improve it later if you want to.)

We’ve used the **new** keyword a few times, so make sure you’ve got some **delete**s in your code! I’ve put mine in my destructor for now.

A screen shot of a computer program

Description automatically generated

## Randomising animal and passport textures

For this, we’re just going to use the random number generation method that we used last year to generate a pair of integers between 0 and 2. If the integers are the same, then we know that the animal and passport match and the player should accept the animal’s entry into the country. If the numbers are different, they’ve given you the wrong passport and you can’t let them in! (This is why the order was important when loading the textures.)

I’ve created a function called newAnimal() that generates a new animal/passport pair for me and sets the textures of the sprites accordingly. Note the 2nd argument in setTexture(), this is an optional argument (which defaults to false) that tells the function whether to re-size the sprite to match the texture. Without it, the textures can end up cropped when we try to fit them into a sprite that’s too small for them.

A computer screen shot of a program code

Description automatically generated

Make sure you’re rendering your character and your passport and calling your newAnimal() function somewhere sensible, then boot up your game! It should show you a randomised pair of animals and passports each time you boot it up.



# dragging your passport

## drag function

Before we start, let’s set up a sprite pointer in our game.h that we can use to point to whatever sprite is being dragged by the mouse. This will let us move any sprite we like just by reassigning our pointer.



Next, we want a function that takes whatever a sprite pointer is pointing to and moves it around with the mouse, with a guard to make sure that the pointer isn’t null. Make sure you call this function each frame so it will move the sprite appropriately.

A screen shot of a computer code

Description automatically generated

(drag\_offset is something I use to make sure I can drag the sprite from wherever I click it, rather than its top corner being attached to the mouse. I’ll let you figure out how to define drag\_offset if you want it.)

## mouse input

So far, we’ve been using the mouse rather simply, assuming that whenever a click happens, it’s the left mouse button and ignoring whether or not the button is being held.

You will remember from last year that SFML has **KeyPressed** and **KeyReleased** events that contain information about which key it is being pressed or released, and our input logic works from that. Similarly, SFML also has **MouseButtonPressed** and **MouseButtonReleased** events that we can use in the same way.

A screen shot of a computer program

Description automatically generated

Now, whenever the player clicks in the game, we can check whether it’s the left mouse button. If it is the left mouse button, and they’re clicking on the passport, we want to start dragging the passport around the screen.

A screen shot of a computer code

Description automatically generated

When the left mouse button is released, just set dragged to nullptr again to stop dragging the passport around.

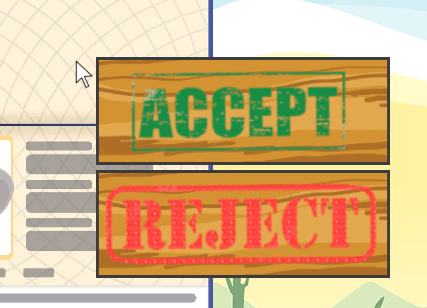
# stamping & returning the passport

Here’s where the guidance gets really light, because most of the stuff we have to do now is directly related to stuff you’ve done before, so you can figure more things out on your own.

## stamping the passport

You’ll want to set up sprites for the accept and reject buttons that are in the images folder, as well as the accept and reject stamps.

Next, you want to use your right mouse button to toggle the appearance of the accept/reject buttons, like your normal Windows context menu.



When your player clicks on one of those sprites, you can set either a passport\_accepted or a passport\_rejected Boolean to true, and then use those Booleans to control whether the accepted/rejected sprites are rendered onto the passport.

Of course, the stamps won’t stay attached to the passport all by themselves, you’re going to have to update the position of any rendered stamp so that it stays the same relative to the passport while your player drags it around…

We *could* create different textures for stamped/unstamped passports and swap the textures when we need to, but this way you get to practice moving objects relative to one another, which is important in many games.

## returning the passport

Now that the passport has a stamp on it, you can return it to the animal that’s waiting in the corner.

The way to do this is to check the position of the passport when the player stops dragging it. If the passport is in the top left corner and it’s been stamped, stop rendering it and let the player know whether they made the right choice to accept/reject that animal. (Remember the should\_accept Boolean from earlier, when we generated the animal/passport pair.)

You can then generate a new animal/passport pair and render them, same as before. Make sure your passport doesn’t arrive stamped.

# closing the game loop

You should already have a menu set up from the whack-a-mole worksheet last week, so we just need a way for the player to lose and boot them back to that main menu, where they can decide whether they want to play again.

You can do this any way you like, I’d start off with a lives counter so they can only make a certain number of mistakes, but after that you can add whatever you like. Timers, belligerent animals, quotas, etc. Do whatever you think will be fun.

With that done, you should have a bare-bones, blatant rip-off of Papers Please, the original of which you can find at the link below:

<https://papersplea.se/>

[App Store Link](https://apps.apple.com/us/app/papers-please/id935216956)

[Google Play Link](https://play.google.com/store/apps/details?id=com.llc3909.papersplease&hl=en_GB)