SIT102 Introduction to Programming

Credit Task 5.3: Game Project (Part 1)

Overview

Great to see you here. I've got an idea for a game and I think it would be great for us to work together on this. Lets get started with an overview of my idea and you can help me build this out into a small game.

Submission Details

Submit the following files to OnTrack.

- A screen shot of your program running
- Your program code (All header files (2) and source code files (3))

The focus of this task is on bringing together what we have done so far in order to create an interactive program. While this is a game, the same skills underlie any programming project.

Instructions

Okay, I've been watching Lost In Space on Netflix and thought it would be fun to create a space game where the user explores a game universe moving between planets, dodge things, collecting coins, getting power-ups, etc. Perhaps there is also a time limit where they are aiming to find a destination planet, with limited fuel. This way there is the trade off between reading the goal and improving your score.

Anyway, that is some of my ideas. To get things going I've created some starter code and there is a video talking though this. Download the task resources and review the code while watching the video. Then it's over to you. I have some things I want you to get working this week. We will keep building on this in the future as well!

Let's get started.

1. Download the starter code in OnTrack resources.zip, add it into a new project on your machine, then compile and run. Havea look around :) hopefully you can see the potential.
2. Watch the Lost In Space Game Code video and use it to get an overview of how the program works, and what the code does.
3. If you need more perspectives on Game Part 1 design and a tasksheet walkthrough* as a start up, you may watch SIT102 Game Part 1 Elaboration. It includes elaboration on the resource, and a walkthrough of most of the contents in the tasksheet.

   * Please note, the tasksheet walkthrough intends to provide you with a smooth start up for the entire Game Project, so it will only be provided in Part 1 (5.3C). You are suggested to start your Game Project Part 2 (7.2C) as soon as possible once your have submitted this task.

Your Task

Now make the following updates to the program code:

1. I was a bit lazy in creating the player… please put them in the centre of the screen (not slightly offset as they are). For this step you will need to update the code in `new_player`.

   Hint: The problem is that the player x, y are based on the top left of the player. One way to solve this is to use the screen_width and sprite_width functions. The correct left side for the player should be the screen width - the sprite width all divided by 2.

   To call **sprite_width** you will need to pass it the sprite to get the width of. If you look at the program's structs you can see that the **player_data** has a **player_sprite** field. This will contain the sprite you want to get the width of. In this case the **result** variable is your **player_data**, so **result.player_sprite** will be the sprite you need to access.

2. Looks like I missed one of the ships… pressing 1, 2, or 3 should switch the player ship. Please fix this so we can switch between the different ships.

   Hint: Start at **handle_input** and step through the code yourself. Where are these changing? Why are they not all working?

3. There are some power up images in the resources folder (I've listed them below). We will use the *Fuel* power-up and I want you to pick another 5 of these - as I don't think we want all of them - and start to add them into the game.

   Note: We will start by getting these showing up as things for the player to collect. Then we can work on effects of these later.

   The power up images include: shield, power, potion, fuel, drops, diamond, star , cash, nos, level, bullet, coin, battery, heart, muscle, and time. They are already loaded, so you can access them by name straight away.

   Please add code to manage these in the project. This will involve:

   - Create a new **power_up.cpp** file, and a new **power_up.h** header file, to manage the power up related code.

     Hint: Remember to add the **header guard** to the header file. Just as you did in the earliertasks.

   - Create an **enum** in the header file to represent these different kinds of power up. We can use this to represent the different power-ups within the code.

- Create a function to convert an power-up enum values to an image.

  Hint: You will need a **switch** based on the passed in parameter, then you can return the bitmap you get back from calling **bitmap_named**. eg. `case SHIELD: return bitmap_named("shield");`.

  Have a look at the names in the **lost_in_space.txt** bundle file.

- Create a struct in the header file to represent a power-up. Each power-up needs to have:
  - The kind of power-up it is
  - A sprite for their visual representation
- Add functions and procedures to work with power-ups: initialise a power-up (call it **new_power_up**) **draw_power_up** and **update_power_up**. Include these in the header and implement in the source file.
  - Implement new power-up creation as **power_up_data new_power_up(double x, double y)**
    - Randomly pick a power_up kind using `static_cast<power_up_kind>(rnd(6))`
    - Use the enum values to pick its bitmap. For example, you can assign `result.power_up_sprite = create_sprite( power_up_bitmap(kind) );`
    - Position the power_up at the indicated x,y location (using the value passed to the related parameters)
    - Start power_ups with a small random velocity.

      Hint: You can do this by calling **sprite_set_dx** - pass in your sprite and a small value for the “change in x”. eg: `sprite_set_dx(result.power_up_sprite, rnd() * 4 - 2);` Here the `rnd()` call will return a random value between 0 and 1. Multiplying this by 4 gives 0 - 4. Subtracting 2 gives -2 to 2.
  - The **draw_power_up** procedure will accept a **power_up_data** value (by constant reference) and can call **draw_sprite** on the **power_up_sprite**.
  - Similarly, **update_power_up** will also accept a **power_up_data** value (this time by reference) and will call **update_sprite** on the **power_up_sprite**. This will move the power_up based on the velocity above.
- Add a power-up into main - start it near the player
  - Make sure to include the new power-up header
  - Remove the rectangle - it was just there so we could see the camera moving.
  - Use **new_power_up(…)** to initialise your power-up - pass in any value for x and y just make it something that will appear on screen.
  - Update the game loop so that it will update and draw the power-up.

4. Improve the heads up display (HUD) - which currently just displays the score and location as text. You can adjust the aspect ratio of the window if you want - it doesn't need to remain the size it is.

    - Create a procedure to draw this, in the *program.cpp* file.
    - Design your own HUD to display the following:
        - Use an image or some shapes to define the area of your HUD. Make it look good if you can (though we will focus on the code).
        - Location of the player
        - Score of the player
        - Area for a minimap - make it 100 pixels by 100 pixels. It should have a black background, on top of which we will later in Game Project (Part 2) to draw the minimap by displaying the player and power-up coordinations.
        - A fuel gauge - using one of the bars from the resources (see below)
        - Then add at least two other features yourself. Think of something that links to your selected power-ups.

    Important: When you implement the code for the HUD you will need to use option to screen so that the coordinates don't move with the camera. Have a look at the article on using the SplashKit camera.

5. Implement the code to draw the background of your HUD.

6. Add the code to show the player's location in your HUD.

7. Add a **fuel_pct** field to the **player** struct, and initialise this to 0.75 when the player is created. Then add the code to draw the fuel gauge on the HUD. This will visually show how much fuel is remaining as a partly full bar.

    There are some "bar" images in the Resources. You can use these to show how much fuel is remains (and maybe for some of your own features on the HUD).

    - Add the bars you want to the resource bundle so they are loaded when the program loads.

    - Draw one of the empty bars to the screen as part of the HUD - or embed it in your background.

    - Over the top of this, draw **part** of one of the full bar bitmaps (your choice which)

      Use draw bitmap with option part bitmap to draw half of the "full" bar. Set the part to start at the 0,0 of the image (top left) use all of its height and 25% (0.25) of its width. Later we can dynamically change this percent, and it should have the effect of making the bar appear to reduce/increase.

      Note: You need to combine the *part option* with the *to screen option* for example: **draw_bitmap("full", bar_x, bar_y, option_part_bmp(0, 0, part_width, bitmap_height("full"), option_to_screen()));**

8. Add two additional visual component for your power-ups, with relevant data values added to the `player_data` struct.

   Ideally the two features should have different visual representations. You could use a bar, show a value, or show something as on/off (grey/coloured). Be creative.

   For each of the additional data items, make sure to initialise them when the player is created and use that value when the HUD is drawn.

Remarks: you have to apply the concepts and skills that had been covered from the previous weeks in your task (including coding convention: comments, indentation, snake_case, and meaningful identifier.).

When done, grab a screenshot of it running and submit to OnTrack. Please leave an audio comment about the project and your new features, or discuss with your tutor in class.

Task Discussion

For this task you will be asked to discuss at least the following:

- Explain the code that you were give. How is it organised, what are the main aspects.
- Discuss your approach to understanding the code. How would you approach larger projects?
- How do the programming artefacts help structure and focus your approach to understanding code?