Task: Task 29
Title: Spike Extension Report
Author: Sam Huffer, 101633177

Instructions

Upload a single document that contains details of all other spike-like extension work that you have carried out and that isn't already covered by an existing spike task or spike report.

Task 4: Gridworld Multi-Threaded

The requirements for this extension task were:

- Create a console program that implements the "Gridworld" game using multi-threading to create a non-blocking game loop. The loop must execute continuously, only processing input when it occurs, and only providing output when necessary. The Gridworld game should be implemented with a timer.

I did not complete this extension as I was prioritising core work but never got back to this task due to my workload between this unit and other units. I would have liked to have completed this extension, though, as multi-threading would be a valuable thing to know how to implement manually.

Task 16: Configuration Files

The requirements for this extension task were:

- Extend your current Zorkish Spike to include an expanded "Adventure" file. Your new file should:
  - Be easy to produce with different values, so you should show at least two different working configurations.
  - Go beyond a graph implementation to include one of the following:
    - Entities (items, containers, etc.)
    - Commands
    - A unique idea you have confirmed with your tutor
- NOTE: If you choose to do this extension, do this AFTER doing the earlier spikes to build on them.

When I went to do this task, I found I had already implemented the first two suggestions (entities and commands) as part of previous tasks. Following consultation with Tien about what I could add for this task, I settled on specifying commands to be locked when the player reached a particular location or added a particular item to their inventory. I put together a UML class diagram to plan what I figured I would need to implement this, updated my text file specification to incorporate this, and updated the code to unlock commands under the aforementioned conditions, with some commands (look, help and quit) always being available regardless of text file specification. For more details, see "Task 16 Extension Report – Configuration Files.docx".

In later tasks I implemented button objects that could be "pressed" to trigger an effect. Were I to update this extension further, I would add functionality such that commands could be unlocked when an associated button is pressed.

Task 19: Messaging Extended

The requirements for this extension task were:

- "Extend" the previous spike to include one or more of the following additional message system features. Update your design documents to reflect the changes of your new system.
- Possible extension features/support:
  - Broadcast messages (specified by the sender)

  - Filtering of messages before delivery/pickup (by the blackboard/dispatch system, not the sender)
  - Scheduling of messages for the future
- You may want (or need) to include the ability for the sender to “delete” or cancel messages in support of the above features.
- Examples for filtering of message could be based on game entity values/types, or locations
- You need to produce:
  - Updated design documents (UML class, module, sequence etc) as applicable, clearly showing what you have had to add to support your additional features, and an
  - Updated working code demonstration within the Zorkish game example.

For this extension task, I decided on implementing a dispatcher-type message system that could broadcast and filter messages. I created a UML class diagram of what I expected I would need. I then implemented the additional dispatcher functionality in several stages, and added the button, landmine and flammable components, and the use command, and changed the open command and associated components to all use and demonstrate different features of the message manager. As I was implementing these, I found that I also needed to implement queuing of messages to allow for deleting destructible game objects outside of loops iterating over data structures containing them. For further details, see Task 19 – Extension Report – Messaging Extended.docx”.

Task 25: Control Mapping

The requirements for this extension task were:

- Create a simple application (that uses your framework (SDL2) capture input events (such as key down or up events), and map the input to a change. You must specifically demonstrate:
  - That input can be mapped to components based on configuration data at run-time
  - Be able to change and reload-input mapping configuration without restarting the application.

I did not complete this extension task, as I was prioritising other tasks required for pass, credit and distinction grades, as well as pending assessments from other units. Had I extra time to focus on this unit, I would like to have completed this task, as remapping controls is a common and expected feature of games.

Task 27: Collisions Extended

The requirements for this extension task were:

- Extend your Collisions spike to display multiple moving entities on screen and add pixel-level tests for collisions. Your application must:
  - Perform collision detection with non-basic entities. Use something more than basic squares or circles. Use at least 2 different sprites.
  - Your sprites should all move at a constant velocity within the screen space
  - Sprites must change colour when colliding so that is obvious that collision has been detected.

I did not complete this extension task, as I was prioritising other tasks required for pass, credit and distinction grades, as well as assessments from other units. Had I extra time to focus on this unit, I would like to have completed this task, as understanding this further level of collision detection would be useful knowledge to have as a game developer. Were I to attempt this task, I would like to set up the pixel-level collision-detection algorithm to test for collision detection at a square-tile-by-square-tile level (perhaps two) before getting to a pixel-level collision test, so as to minimise the number of pixels to be tested. I would think the movement of sprites at a constant velocity would be straightforward, given my previous education in AI for Games, and the sprites changing colour would be straightforward enough, seemingly being just a combination of using the colour-switching code for Task 26 but swapping Texture images instead of changing colours.