Draft Joint Project Briefing [Monday 12th April to Friday 23rd April]

Instructions:

1. Final commits to your github team repository by 8am Friday 23rd April.
2. You will be required to demonstrate your projects in class on Friday 23rd April.

Brief description of gameplay

Create a top down 2D game of your choosing. The game should have a simple menu system where some aspect of the game can be customised (e.g. player avatar). Implement a game system with an appropriate HUD and game states such as game win, lose etc. The game must include an AI that can negotiate its way around impassable areas of the map using a simple pathfinding technique (see Programming specification). The player has the capability to shoot projectiles at the non-player characters, so a collision detection / response system is required. The visual elements of the game should include animated sprites, particle effects and use of shaders.

Finally implement at least one interesting game mechanic. For example, there might be a novel control system for the player movement (sliding, flying, bouncing, dashing or you shoot yourself around like a billiard ball).

Software Engineering specification

1. Use the github “project” feature to create a project board with the following 4 columns “In preparation”, “Ready”, “In Progress”, “Done”.
2. Construct at least 4 user stories for an MVP game. Each story must follow the INVEST and 3Ws guidelines. The stories should be **around core gameplay**. Each story must have acceptance conditions. Add the stores to the board. [20 marks, 5 marks each]
3. Using the stories and any other game requirements, construct CRC cards. The CRC cards should be placed in a text repo file called “crc.md”. Use the vertical text pattern similar to what we used in stormboard. [20 marks]
4. Once the CRC cards are done, used them to produce an initial domain model, showing responsibilities and relationships. This should be placed in a file called “domain_model.md”. The file will contain PlantUML script plus a screenshot of rendered model [20 marks]
5. After the project is complete, produce a class diagram of the final code. This should be placed in a file called “class_dagram.md”. The file will contain PlantUML script plus a high-res screenshot of rendered diagram. This will be due 5pm on Friday 23rd, so try to keep it updated daily. [20 marks]
6. Your final architecture should follow SOLID principles as much as practicable. I will be using the code and class diagram to assess this, so make sure the class diagram is accurate and complete (all attributes, public methods, relationships, and dependencies (including OperatingSystem, Language API and SFML)). [20 marks]

Programming specification (20% weighting for this subject)

1. The game world (map) should be divided up into a grid. Each cell in the grid should store a list of its neighbouring cells. There can be up to eight neighbouring cells assuming movement is possible in all directions. Use the neighbours algorithm to help you generate the list.

[20 marks]

2. Your game world will have areas of the map that are impassable, for example, walls or some other entity that the player/AI cannot walk through. Represent these areas by positioning static sprites such as walls etc. in the appropriate places on your map. You will need to mark any cells in your grid as impassable by reading the world position of all relevant sprites and marking any cells they overlay as impassable.

[20 marks]

3. Implement the breadth-first search algorithm so it can find a route between any two points on the game map (e.g. the AIs current position to another traversable area of the map). This route should be returned as a list of adjacent cells. Implementation note: Consider implementing a Grid class, where a Grid has many Cells (Cell is a separate class). The Grid class might have a member function to perform the breadth-first search:

```
std::vector<int> Grid::breadthFirst(int t_startCellId, int t_destCellId)
```

[30 marks]

4. The AI should find its way from cell to cell until it reaches the desired destination.

[10 marks]

5. Finally, add a debug mode so the 2D grid is drawn and individual cells are shaded showing the progress of the BFS algorithm as it runs. Cells on the final path found should be shaded a different colour as well as cells that represent impassable areas.

[20 marks]

6. Bonus: If you attempt this section, you can "top up" your project mark up to a maximum of 100. See if you can implement an optimised version of the search algorithm. For each cell, include an estimate of its distance to the goal. In the search algorithm, always process the child node that is *closest to the goal* next. Consider using std::priority_queue in place of std::queue.

[20 marks]

Each section will be marked accordingly:

0-1	**Not done/nothing of value or significant problems.**
2-3	**Partial implementation, something of merit, implementation mostly uncommented.**
4-5	**Working to specification, fully commented. May have minor problems.**

Motion graphics specification

Sprite Sheet Animation

[30 Marks]

Collision Detection and Response Projectiles

[10 marks]

Particle Effects

[20 Marks]

Simple Menu System

[20 Marks]

Overall Game System

[20 Marks]

Gameplay Programming I (Joint Project Rubric)

Mark Appropriation: 15% of 100%

Description: produce an SFML Game which adheres to the rubric specification. The title should include visual effect(s) and gameplay mechanic(s).

Team Rubric		
Basic	**Intermediate**	**Advanced**
0 - 5	**5 - 11**	**11 - 15**
• Gameplay Programming requirements have been implemented to a basic level. • Gameplay Programming requirements implementation will achieve minimum functionality. • Gameplay Programming requirements implementation may contain some syntax and/or run-time errors. • Gameplay Programming requirements implementation code will be poorly commented and/or formatted. • Gameplay Programming requirements will not be tested properly.	• Gameplay Programming requirements have been implemented to an acceptable level. • Gameplay Programming requirements implementation will achieve expected functionality. • Gameplay Programming requirements implementation will not contain syntax and/or run-time errors. • Gameplay Programming requirements implementation code will be reasonably commented and/or formatted. • Gameplay Programming requirements will	• Gameplay Programming requirements been implemented to an advanced level. • Gameplay Programming requirements implementation will include novel gameplay. • Gameplay Programming requirements implementation will not contain syntax and/or run-time errors. • Gameplay Programming requirements implementation code will be well commented and/or formatted. • Gameplay Programming requirements will be expertly tested.

• Gameplay Programming requirements implementation code will not follow applicable coding conventions.	be tested to a reasonable degree. • Gameplay Programming requirements implementation code will follow appropriate coding conventions.	• Gameplay Programming requirements implementation of code will follow coding conventions.
Submitted game will contain the following:		
Visual Effect • A GameObject within a gameplay, cut or transition scene will have a visual effect applied which utilised a shader to modify the appearance of that object. • Basic effects include recolouring of GameObjects.	**Visual Effects** • GameObjects within a gameplay, cut or transition scene will have visual effects applied which utilised shaders to modify the appearance of the scene. • Intermediate effects include object processing, lighting and scene colouring.	**Visual Effects** • Scene(s) will have visual effects applied which utilised shaders to modify the appearance of the entire title to achieve visual harmony. • Advanced effects include particles, scene lighting, shadows, scene object processing.
Gameplay Mechanic • A GameObject within a gameplay, cut or transition scene will have a movement mechanic.	**Gameplay Mechanics** • A GameObject within a gameplay, cut or transition scene will have appropriate movement mechanics that 'feels' appropriate to the game.	**Gameplay Mechanics** • A GameObject within a gameplay, cut or transition scene will have appropriate movement mechanics that 'feels' appropriate to the game and in implemented to an advanced level.

	• Implemented to an acceptable level.	• The mechanic maintains gameplay immersion.