Architecture and Design Rationales

Two key classes debated as a team.

GameBoard

The main issue we debated as a team is for the class GameBoard, on how to not make it a god class. In the initial stage of the design, the game board is responsible to

1. check for valid moves,
2. check for which part of the game board is clicked,
3. take care for the interaction between tiles on gameboard, create and destroy them,
4. check for conditions after user's action,
5. handle do & undo feature.

To solve 2), we filled each space of the game board with tiles that is clickable. The gameboard will be notified when its element is clicked. To solve 3), we implemented the methods that handle interaction between tiles in class Tile. To solve 4), we introduce observers to our design. To solve 5), the Command and GameMemento will handle the implementation.

In the end, the game board only needs to check for valid moves. One of the characteristics of a god class is having many methods. As shown in the UML, the class GameBoard has only 1 important public setter method, which is *Notify()*. The game board will be notified when the tiles are clicked, then it will check for valid moves and demand the tiles to interacts with each other.

In future, there may be more interactions between tiles, which will be added as methods in class Tile. More complex situations of valid moves will be subdivided into multiple private methods in class GameBoard. In the end, the main responsibility of game board is to check for valid moves, and the public interface is *Notify().*

GameMemento

In the previous design of GameMemento, we planned for the GameBoard to be able to directly load in a GameMemento like a game snapshot, loading all the state of the game at once including the command history and other relevant information. However, upon revision, we found out that a game can be simply represented by the Commands alone, thus, the design of GameMemento is simplified, we will still be able to load a game purely by replaying the commands on an empty game. It will also have the feature as described by Memento pattern, where the last known state is able to be restored, this is done in combination with Command pattern, by simply reversing (calling the undo() function instead of execute()) the application of Command operation.

Two key relations

Scene to UI (Composition)

A Scene can have many UI elements. Currently, these are Buttons, e.g: The Quit button, the Tiles which are implemented as Buttons, etc. These are visual elements on the screen that can be interacted with. UI Elements and Scenes are closely related because a UI element can only be rendered on a Scene. The relation from Scene to UI also allows Scene to forward Events to UI elements, simulating the intractability of UI elements.

GameBoard to Tiles (Composition)

Tiles is a part of *GameBoard*. Tile is a type of button and is the only way the users can interact with the game board. There are 24 Tiles corresponding to the 24 spaces on the board of a Nine Men Morris's game. *GameBoard* can't exist without *Tiles*.

Two sets of cardinalities

Gameboard to Tile (1 → 24)

A game board has 24 tiles because there are 24 spaces to place a token. In our implementation, the gameboard will be always full of tiles. This is because vacant spaces are clickable as tokens; therefore, we see it as "invisible token" (I.e: No token is on it). As such, we only need to change the render state of the token based on user's action. In the code, we don't need swap, remove, and add tokens to the gameboard, just swap the "state" of that Tile, which makes the implementation easier. This creates abstraction to user that he is adding, removing, or adding tokens to the gameboard. For illustration purpose, the vacant spaces are rendered as grey rectangles in the sprint 2 submission game.

*Game to Scene (1 → 1 … *)*

The game should at least have 1 scene so that users can interact with it. This is so that we can have a different number of different "Scenes" (A screen view with UI elements, so we will have a Scene for the Main Menu, one for the Game itself, HighScore, etc.) that we can switch over. We enforce a minimum of 1 Scene because if there isn't any Scene, the Game class would not be doing anything at all and would just exit immediately.

Explain your decisions around inheritance, why did you decide to use (or not use) it? Why is your decision justified in your design?

Classes sf::Transformable is an SFML base class that defines something that can be transformed, and sf::Drawable is something can be drawn to a RenderTarget (e.g: A Window).

During the designing of UI abstract class, the inheritance of sf::Transformable, sf::Drawable against sf::Shape which inherits both of the former classes was considered. In the end, we decided that the functionality of a Shape is not required.
A Shape in SFML's definition can represent a bounding base shape. It can be convenient for us to inherit Shape in UI and have the Interactable area calculated by EventListener, mainly OnClickEventListener obtain the bounding shape's exact size easily.
However, our definition of UI is any compound SFML elements such as RectangleShape or Text, if we would want the UI to be only partially interactable, for instance a RectangleShape followed by a Text beside it but only the RectangleShape is interactable (Like a radio-box or a checkbox), it is not possible as a bounding shape will enclose all the area of a UI.

A scene switching mechanism is implemented in the main Game, as such, the Game is a Scene player. A layer of abstraction is added to the Scenes, as there can be multiple scenarios a Game can be in, the main Game scene, the main menu for instance. For any simple game, directly adding all the UI and things to the Game is usually done, considering that can potentially make Game contain all the specific scenarios' layout and logic, it can become messy. And hence each scenario is made into Scenes, which implement an Update() function amd Render() function, which the Game simply needs to select one to be active at once and leave the layout (drawing/rendering) and logic (updating) down to the scene. All distinct scenarios must inherit Scene to be shown by the Game, this delegates the layout design and logic to the individual scene and not the Game's.

The class Tile inherits the class Button because Tiles has the behaviours of a pattern. Tiles are just buttons without text.

Explain why you have applied a particular design pattern for your software architecture?
Explain why you ruled out two other feasible alternatives?

In the design of UI having EventListeners, we employ the Observer Pattern. Which allows our UI to be interactable while hiding the implementation details of handling of interaction events by each UI. Once we create the UI and registered to the Event emitter, in this case it is passed down from the Update() function to pollEvents, we do not need to handle the Events for each Interactable element we add explicitly, which can quickly clutter the Scene Update() function, and it is prone to errors.

There exist other designs as well that we have taken into consideration before deciding to go for Observer pattern.

One such pattern is Actor Model, in which everything is an Actor, and they can communicate through a channel such as MessageQueues. This can allow easy communication from elements for instance a Button to interact and send a message to GameBoard, decoupling the dependency from the interact action of the button to gameboard.
However, this design overcomplicates our requirements, as a decoupled communication channel is not required for every object, and the added complexity that each actor must handle multiple types of messages. While this may be an advantage for the clarity of how each message is passed and how each message is handled to the Actor, it overcomplicates the implementation as that means that messages like call to move from a Player is handled in the gameboard, risking it to become a God class.

Another pattern is Mediator pattern, which is a separate communication object encapsulating the communication logic. This slightly resolves the issue of being a God Class, as they are delegating the communication elsewhere, but the problem of handling of multiple types of messages, in Mediator object persists. This pattern is advantageous when all objects that communicate have the same context, but this is not the case in this design, we are sending message from a UI to any element, which that element's mediator may need to handle messages from various sources of distinct types. This makes the disadvantage outweigh the advantage of Mediator pattern.

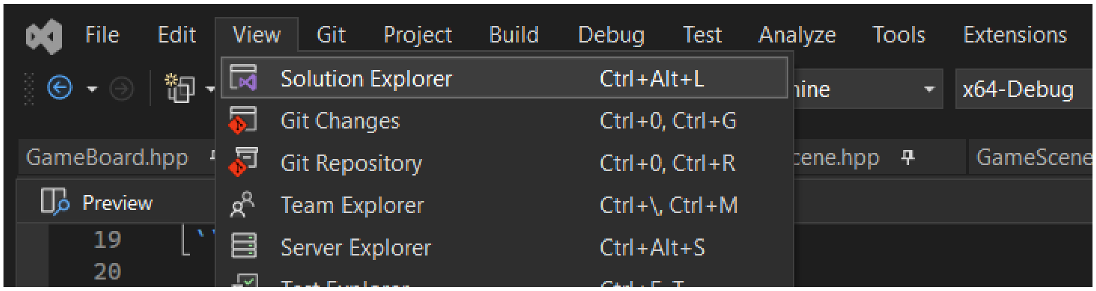
The main argument for Observer pattern to be selected is the ability to set call-back functions listening to the Events, delegating the handling of Events into the call-back function, which is free to perform any functionality it wants to the elements in the scope of the UI.

Visual Studio Graphic Manual (Windows)

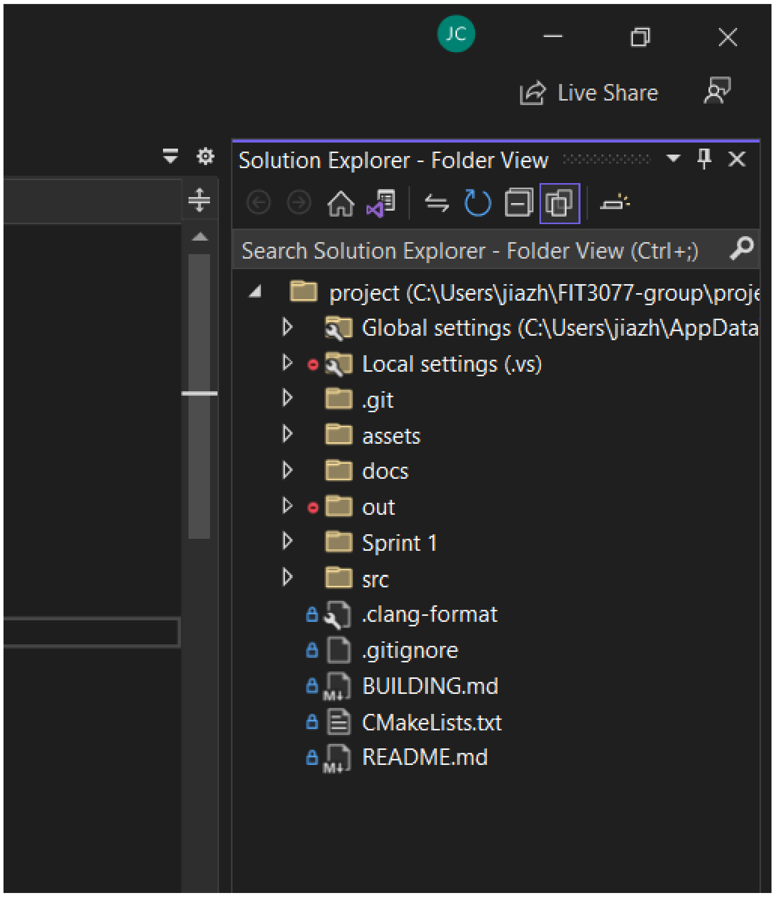
Recommended way to build the project

Please ensure you are using at least Visual Studio C++ 2022

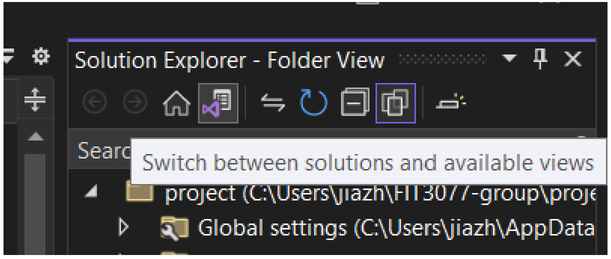
1) Open Solution Explorer

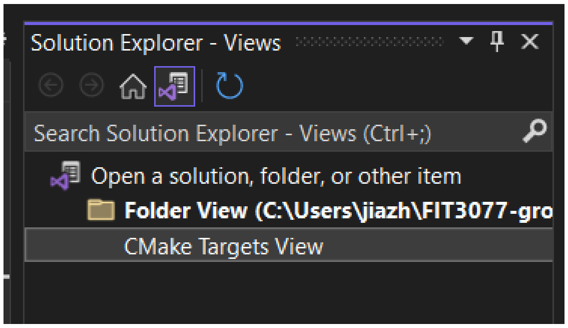

File
Edit
View
Git
Project
Build
Debug
Test
Analyze
Tools
Extensions
Solution Explorer Ctrl+Alt+L
Git Changes Ctrl+0, Ctrl+G
Git Repository Ctrl+0, Ctrl+R
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
x64-Debug
GameBoard.hpp
cene.hpp
GameScene
Preview

It appears as

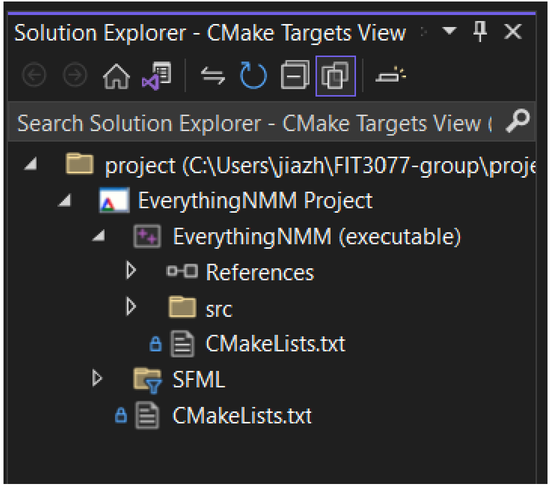

JC
Live Share
Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+;)
project (C:\Users\jiazh\FIT3077-group\proje
Global settings (C:\Users\jiazh\AppData
Local settings (.vs)
.git
assets
docs
out
Sprint 1
src
.clang-format
.gitignore
BUILDING.md
CMakeLists.txt
README.md

2) Switch to CMake Targets View


Solution Explorer - Folder View
Searc
Switch between solutions and available views
project (C:\Users\jiazh\FIT3077-group\proje
Global settings (C:\Users\jiazh\AppData

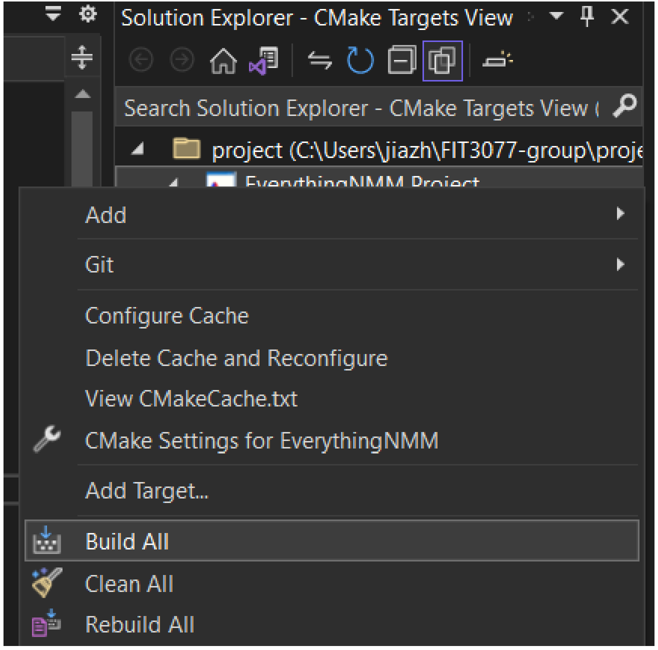

Solution Explorer - Views
Search Solution Explorer - Views (Ctrl+;)
Open a solution, folder, or other item
Folder View (C:\Users\jiazh\FIT3077-gro
CMake Targets View

It appears as

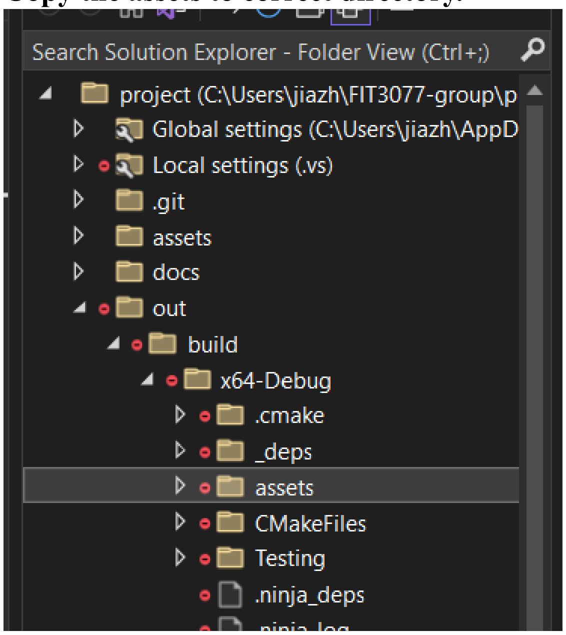

Solution Explorer - CMake Targets View
Search Solution Explorer - CMake Targets View (
project (C:\Users\jiazh\FIT3077-group\proje
EverythingNMM Project
EverythingNMM (executable)
References
src
CMakeLists.txt
SFML
CMakeLists.txt

3) **Build the project.**

Right click *Everything NMM Project* & *Build All*

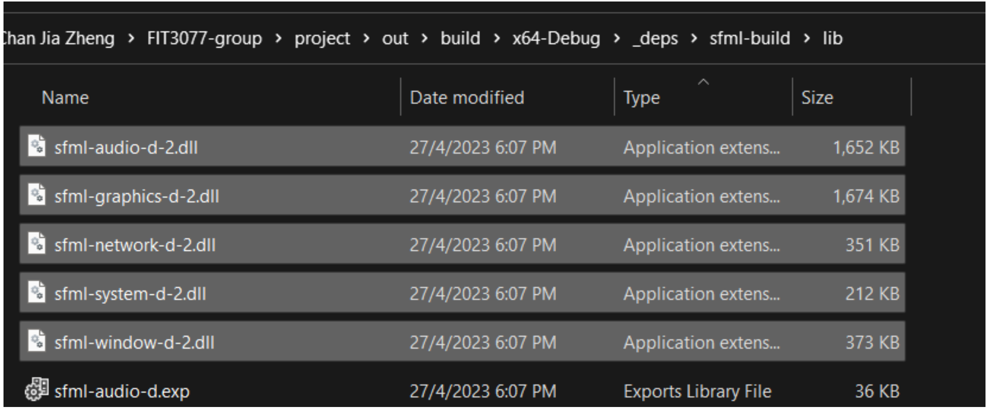

Solution Explorer - CMake Targets View
Search Solution Explorer - CMake Targets View (
project (C:\Users\jiazh\FIT3077-group\proje
EverythingNMM Project
Add
Git
Configure Cache
Delete Cache and Reconfigure
View CMakeCache.txt
CMake Settings for EverythingNMM
Add Target...
Build All
Clean All
Rebuild All

4) **Copy the assets to correct directory.**


Search Solution Explorer - Folder View (Ctrl+;)
project (C:\Users\jiazh\FIT3077-group\p
Global settings (C:\Users\jiazh\AppD
Local settings (.vs)
.git
assets
docs
out
build
x64-Debug
.cmake
_deps
assets
CMakeFiles
Testing
.ninja_deps
.ninja_log

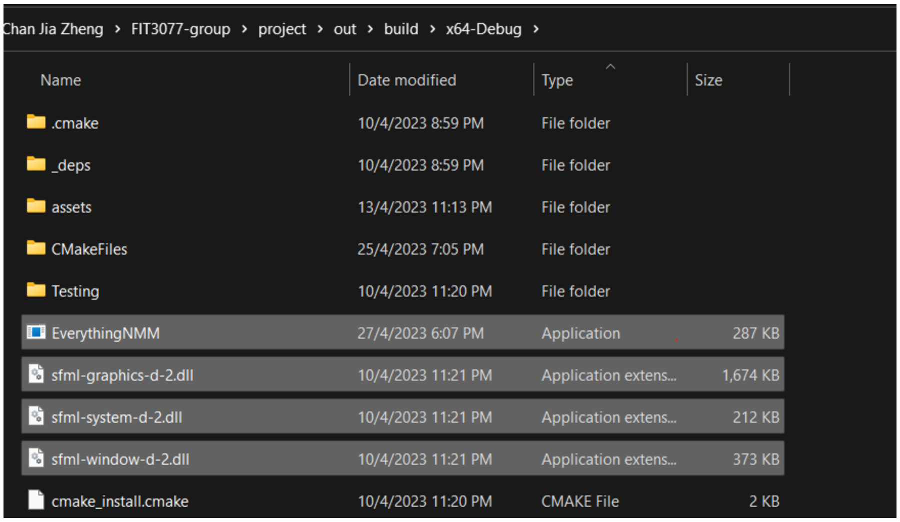
Copy folder *assets* to ***./out/build/x64-Debug/***

5) **Copy the .dll files to correct directory.**
./out/build/<BUILD_TYPE>/_deps/sfml-build/lib/ to ./out/build/<BUILD_TYPE>/



Chan Jia Zheng › FIT3077-group › project › out › build › x64-Debug › _deps › sfml-build › lib

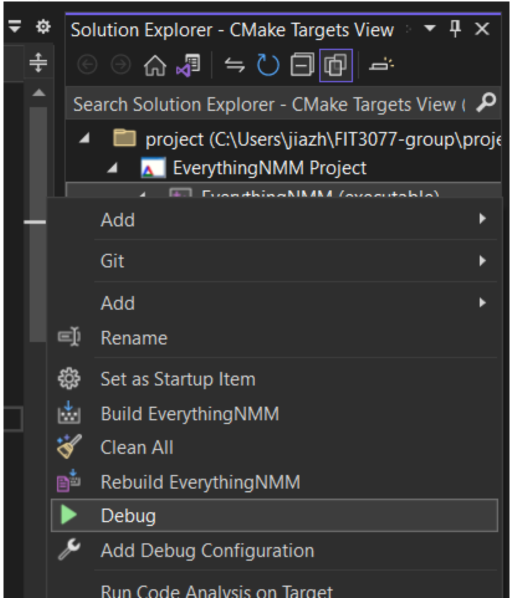
Name	Date modified	Type	Size
sfml-audio-d-2.dll	27/4/2023 6:07 PM	Application extens...	1,652 KB
sfml-graphics-d-2.dll	27/4/2023 6:07 PM	Application extens...	1,674 KB
sfml-network-d-2.dll	27/4/2023 6:07 PM	Application extens...	351 KB
sfml-system-d-2.dll	27/4/2023 6:07 PM	Application extens...	212 KB
sfml-window-d-2.dll	27/4/2023 6:07 PM	Application extens...	373 KB
sfml-audio-d.exp	27/4/2023 6:07 PM	Exports Library File	36 KB



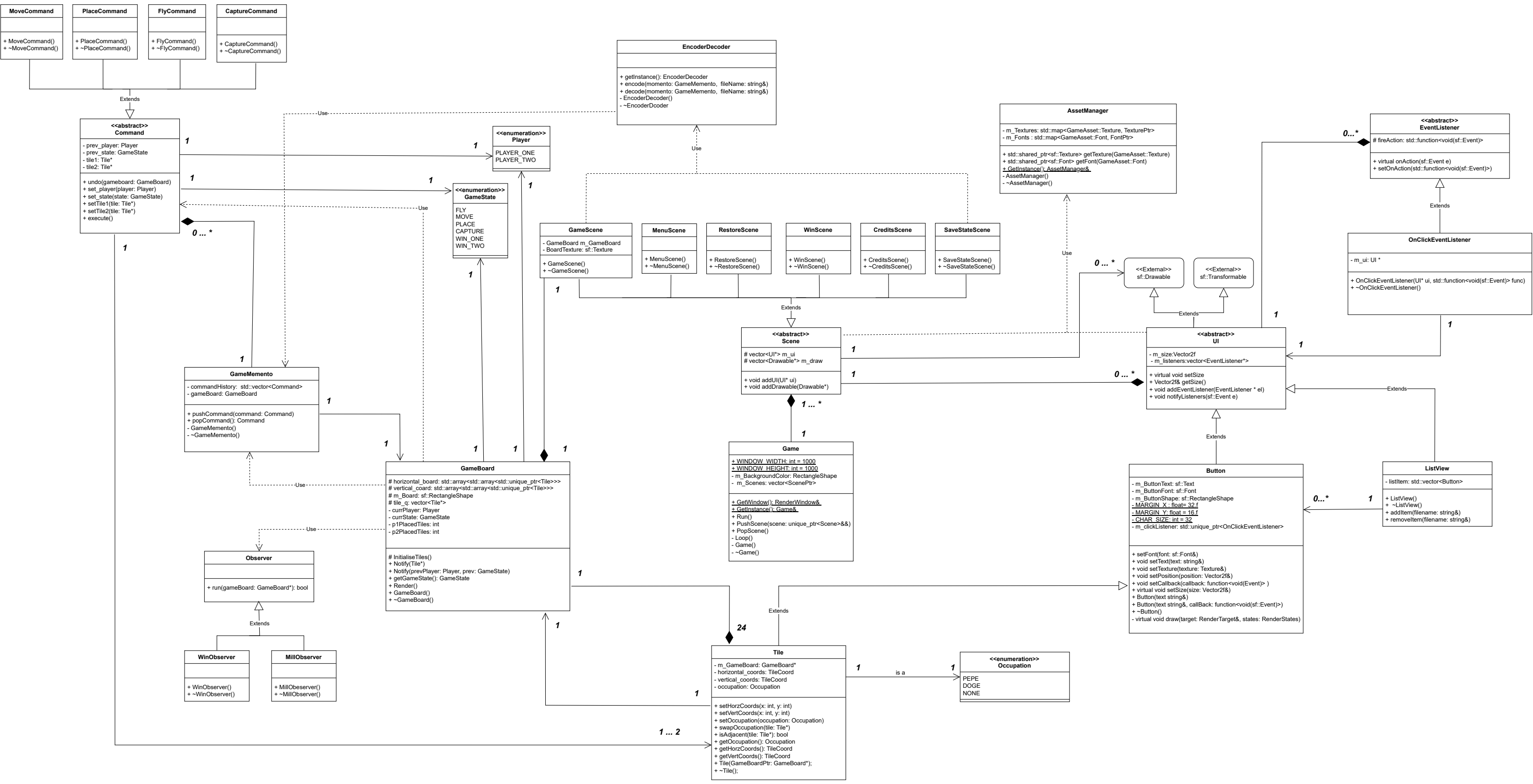
Chan Jia Zheng › FIT3077-group › project › out › build › x64-Debug ›

Name	Date modified	Type	Size
.cmake	10/4/2023 8:59 PM	File folder	
_deps	10/4/2023 8:59 PM	File folder	
assets	13/4/2023 11:13 PM	File folder	
CMakeFiles	25/4/2023 7:05 PM	File folder	
Testing	10/4/2023 11:20 PM	File folder	
EverythingNMM	27/4/2023 6:07 PM	Application	287 KB
sfml-graphics-d-2.dll	10/4/2023 11:21 PM	Application extens...	1,674 KB
sfml-system-d-2.dll	10/4/2023 11:21 PM	Application extens...	212 KB
sfml-window-d-2.dll	10/4/2023 11:21 PM	Application extens...	373 KB
cmake_install.cmake	10/4/2023 11:20 PM	CMAKE File	2 KB

6) **Execute the game.**
Right click *EverythingNMM (executable)* & *Debug / Run the EXE file.*


Solution Explorer - CMake Targets View
Search Solution Explorer - CMake Targets View (
project (C:\Users\jiazh\FIT3077-group\proje
EverythingNMM Project
EverythingNMM (executable)
Add
Git
Add
Rename
Set as Startup Item
Build EverythingNMM
Clean All
Rebuild EverythingNMM
Debug
Add Debug Configuration
Run Code Analysis on Target

UML Class Diagram


MoveCommand
+ MoveCommand()
+ ~MoveCommand()
PlaceCommand
+ PlaceCommand()
+ ~PlaceCommand()
FlyCommand
+ FlyCommand()
+ ~FlyCommand()
CaptureCommand
+ CaptureCommand()
+ ~CaptureCommand()
Extends
<<abstract>>
Command
- prev_player: Player
- prev_state: GameState
- tile1: Tile*
- tile2: Tile*
+ undo(gameboard: GameBoard)
+ set_player(player: Player)
+ set_state(state: GameState)
+ setTile1(tile: Tile*)
+ setTile2(tile: Tile*)
+ execute()
<<enumeration>>
Player
PLAYER_ONE
PLAYER_TWO
<<enumeration>>
GameState
FLY
MOVE
PLACE
CAPTURE
WIN_ONE
WIN_TWO
EncoderDecoder
+ getInstance(): EncoderDecoder
+ encode(momento: GameMemento, fileName: string&)
+ decode(momento: GameMemento, fileName: string&)
- EncoderDecoder()
- ~EncoderDcoder
AssetManager
- m_Textures: std::map<GameAsset::Texture, TexturePtr>
- m_Fonts : std::map<GameAsset::Font, FontPtr>
+ std::shared_ptr<sf::Texture> getTexture(GameAsset::Texture)
+ std::shared_ptr<sf::Font> getFont(GameAsset::Font)
+ GetInstance(): AssetManager&
- AssetManager()
- ~AssetManager()
<<abstract>>
EventListener
# fireAction: std::function<void(sf::Event)>
+ virtual onAction(sf::Event e)
+ setOnAction(std::function<void(sf::Event)>)
OnClickEventListener
- m_ui: UI *
+ OnClickEventListener(UI* ui, std::function<void(sf::Event)> func)
+ ~OnClickEventListener()
GameScene
- GameBoard m_GameBoard
- BoardTexture: sf::Texture
+ GameScene()
+ ~GameScene()
MenuScene
+ MenuScene()
+ ~MenuScene()
RestoreScene
+ RestoreScene()
+ ~RestoreScene()
WinScene
+ WinScene()
+ ~WinScene()
CreditsScene
+ CreditsScene()
+ ~CreditsScene()
SaveStateScene
+ SaveStateScene()
+ ~SaveStateScene()
<<External>>
sf::Drawable
<<External>>
sf::Transformable
<<abstract>>
Scene
# vector<UI*> m_ui
# vector<Drawable*> m_draw
+ void addUI(UI* ui)
+ void addDrawable(Drawable*)
<<abstract>>
UI
- m_size:Vector2f
- m_listeners:vector<EventListener*>
+ virtual void setSize
+ Vector2f& getSize()
+ void addEventListener(EventListener * el)
+ void notifyListeners(sf::Event e)
GameMemento
- commandHistory: std::vector<Command>
- gameBoard: GameBoard
+ pushCommand(command: Command)
+ popCommand(): Command
- GameMemento()
- ~GameMemento()
GameBoard
# horizontal_board: std::array<std::array<std::unique_ptr<Tile>>>
# vertical_coard: std::array<std::array<std::unique_ptr<Tile>>>
# m_Board: sf::RectangleShape
# tile_q: vector<Tile*>
- currPlayer: Player
- currState: GameState
- p1PlacedTiles: int
- p2PlacedTiles: int
# InitialiseTiles()
+ Notify(Tile*)
+ Notify(prevPlayer: Player, prev: GameState)
+ getGameState(): GameState
+ Render()
+ GameBoard()
+ ~GameBoard()
Game
+ WINDOW_WIDTH: int = 1000
+ WINDOW_HEIGHT: int = 1000
- m_BackgroundColor: RectangleShape
- m_Scenes: vector<ScenePtr>
+ GetWindow(): RenderWindow&
+ GetInstance(): Game&
+ Run()
+ PushScene(scene: unique_ptr<Scene>&&)
+ PopScene()
- Loop()
- Game()
- ~Game()
Button
- m_ButtonText: sf::Text
- m_ButtonFont: sf::Font
- m_ButtonShape: sf::RectangleShape
- MARGIN_X : float= 32.f
- MARGIN_Y: float = 16.f
- CHAR_SIZE: int = 32
- m_clickListener: std::unique_ptr<OnClickEventListener>
+ setFont(font: sf::Font&)
+ void setText(text: string&)
+ void setTexture(texture: Texture&)
+ void setPosition(position: Vector2f&)
+ void setCallback(callback: function<void(Event)>)
+ virtual void setSize(size: Vector2f&)
+ Button(text string&)
+ Button(text string&, callBack: function<void(sf::Event)>)
+ ~Button()
- virtual void draw(target: RenderTarget&, states: RenderStates)
ListView
- listItem: std::vector<Button>
+ ListView()
+ ~ListView()
+ addItem(filename: string&)
+ removeItem(filename: string&)
Observer
+ run(gameBoard: GameBoard*): bool
WinObserver
+ WinObserver()
+ ~WinObserver()
MillObserver
+ MillObeserver()
+ ~MillObeserver()
Tile
- m_GameBoard: GameBoard*
- horizontal_coords: TileCoord
- vertical_coords: TileCoord
- occupation: Occupation
+ setHorzCoords(x: int, y: int)
+ setVertCoords(x: int, y: int)
+ setOccupation(occupation: Occupation)
+ swapOccupation(tile: Tile*)
+ isAdjacent(tile: Tile*): bool
+ getOccupation(): Occupation
+ getHorzCoords(): TileCoord
+ getVertCoords(): TileCoord
+ Tile(GameBoardPtr: GameBoard*);
+ ~Tile();
<<enumeration>>
Occupation
PEPE
DOGE
NONE
is a
Use
Extends
1
0 ... *
0...*
1 ... *
1 ... 2
24