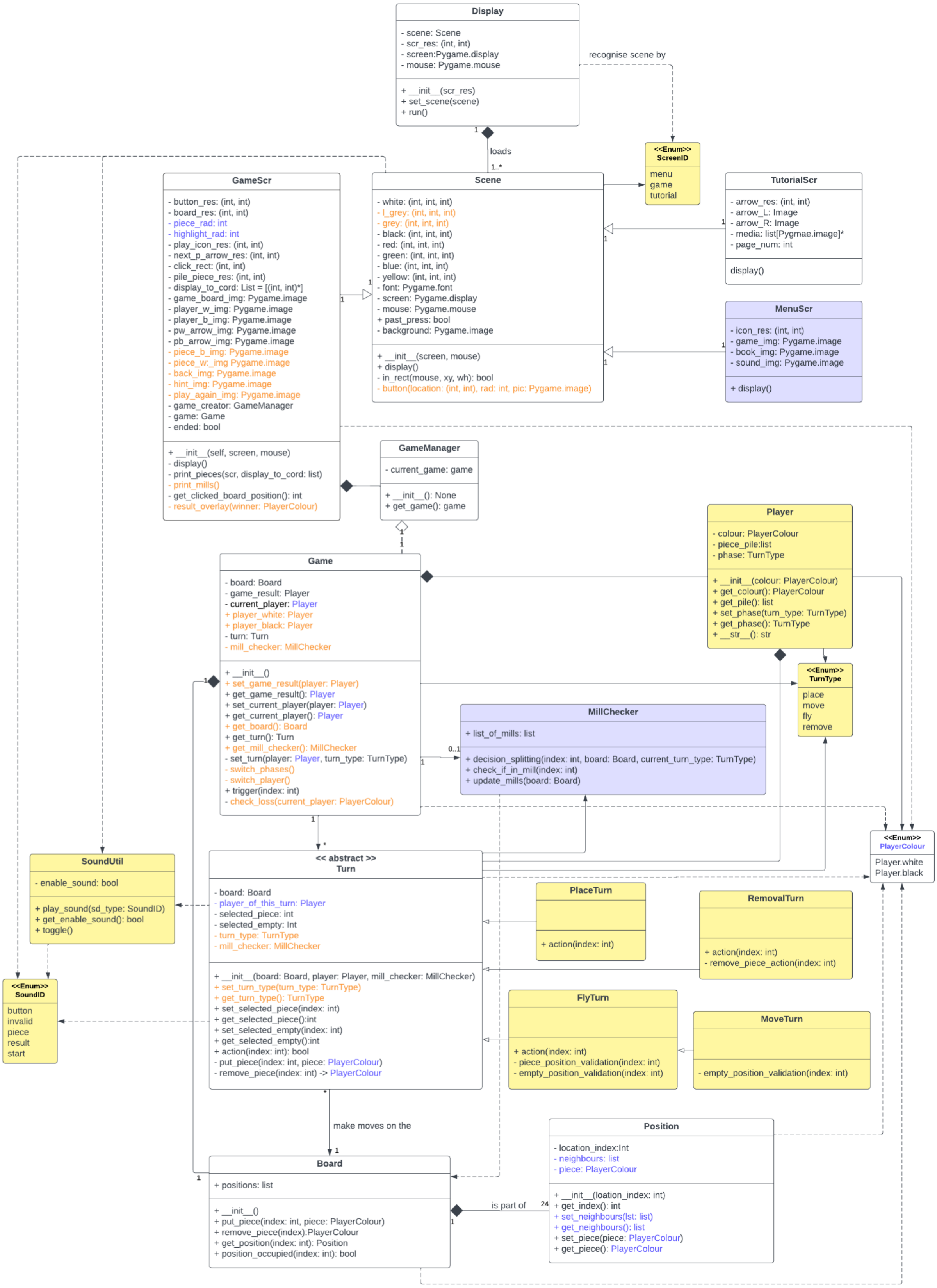
FIT3077 - Software engineering: Architecture and design

Sprint 3 - Rationale

CL_Monday6pm_Team37

Team: HTML is an OOP Language

Class Diagram | New stuff in yellow, Changes in blue


Display
- scene: Scene
- scr_res: (int, int)
- screen:Pygame.display
- mouse: Pygame.mouse
+ __init__(scr_res)
+ set_scene(scene)
+ run()
recognise scene by
1
loads
1..*
<<Enum>>
ScreenID
menu
game
tutorial
GameScr
- button_res: (int, int)
- board_res: (int, int)
- piece_rad: int
- highlight_rad: int
- play_icon_res: (int, int)
- next_p_arrow_res: (int, int)
- click_rect: (int, int)
- pile_piece_res: (int, int)
- display_to_cord: List = [(int, int)*]
- game_board_img: Pygame.image
- player_w_img: Pygame.image
- player_b_img: Pygame.image
- pw_arrow_img: Pygame.image
- pb_arrow_img: Pygame.image
- piece_b_img: Pygame.image
- piece_w:_img Pygame.image
- back_img: Pygame.image
- hint_img: Pygame.image
- play_again_img: Pygame.image
- game_creator: GameManager
- game: Game
- ended: bool
+ __init__(self, screen, mouse)
- display()
- print_pieces(scr, display_to_cord: list)
- print_mills()
- get_clicked_board_position(): int
- result_overlay(winner: PlayerColour)
1
1
Scene
- white: (int, int, int)
- l_grey: (int, int, int)
- grey: (int, int, int)
- black: (int, int, int)
- red: (int, int, int)
- green: (int, int, int)
- blue: (int, int, int)
- yellow: (int, int, int)
- font: Pygame.font
- screen: Pygame.display
- mouse: Pygame.mouse
+ past_press: bool
- background: Pygame.image
+ __init__(screen, mouse)
+ display()
- in_rect(mouse, xy, wh): bool
- button(location: (int, int), rad: int, pic: Pygame.image)
1
1
1
1
TutorialScr
- arrow_res: (int, int)
- arrow_L: Image
- arrow_R: Image
- media: list[Pygmae.image]*
- page_num: int
display()
MenuScr
- icon_res: (int, int)
- game_img: Pygame.image
- book_img: Pygame.image
- sound_img: Pygame.image
+ display()
GameManager
- current_game: game
+ __init__(): None
+ get_game(): game
1
1
Player
- colour: PlayerColour
- piece_pile:list
- phase: TurnType
+ __init__(colour: PlayerColour)
+ get_colour(): PlayerColour
+ get_pile(): list
+ set_phase(turn_type: TurnType)
+ get_phase(): TurnType
+ __str__(): str
Game
- board: Board
- game_result: Player
- current_player: Player
+ player_white: Player
+ player_black: Player
- turn: Turn
- mill_checker: MillChecker
+ __init__()
+ set_game_result(player: Player)
+ get_game_result(): Player
+ set_current_player(player: Player)
+ get_current_player(): Player
+ get_board(): Board
+ get_turn(): Turn
+ get_mill_checker(): MillChecker
- set_turn(player: Player, turn_type: TurnType)
- switch_phases()
- switch_player()
+ trigger(index: int)
- check_loss(current_player: PlayerColour)
1
<<Enum>>
TurnType
place
move
fly
remove
MillChecker
+ list_of_mills: list
+ decision_splitting(index: int, board: Board, current_turn_type: TurnType)
+ check_if_in_mill(index: int)
+ update_mills(board: Board)
0..1
1
1
*
<<Enum>>
PlayerColour
Player.white
Player.black
SoundUtil
- enable_sound: bool
+ play_sound(sd_type: SoundID)
+ get_enable_sound(): bool
+ toggle()
<< abstract >>
Turn
- board: Board
- player_of_this_turn: Player
- selected_piece: int
- selected_empty: Int
- turn_type: TurnType
- mill_checker: MillChecker
+ __init__(board: Board, player: Player, mill_checker: MillChecker)
+ set_turn_type(turn_type: TurnType)
+ get_turn_type(): TurnType
+ set_selected_piece(index: int)
+ get_selected_piece():int
+ set_selected_empty(index: int)
+ get_selected_empty():int
+ action(index: int): bool
- put_piece(index: int, piece: PlayerColour)
- remove_piece(index: int) -> PlayerColour
PlaceTurn
+ action(index: int)
RemovalTurn
+ action(index: int)
- remove_piece_action(index: int)
<<Enum>>
SoundID
button
invalid
piece
result
start
FlyTurn
+ action(index: int)
- piece_position_validation(index: int)
- empty_position_validation(index: int)
MoveTurn
- empty_position_validation(index: int)
*
make moves on the
1
Position
- location_index:Int
- neighbours: list
- piece: PlayerColour
+ __init__(loation_index: int)
+ get_index(): int
+ set_neighbours(lst: list)
+ get_neighbours(): list
+ set_piece(piece: PlayerColour)
+ get_piece(): PlayerColour
1
Board
+ positions: list
+ __init__()
+ put_piece(index: int, piece: PlayerColour)
+ remove_piece(index):PlayerColour
+ get_position(index: int): Position
+ position_occupied(index: int): bool
is part of
24
1

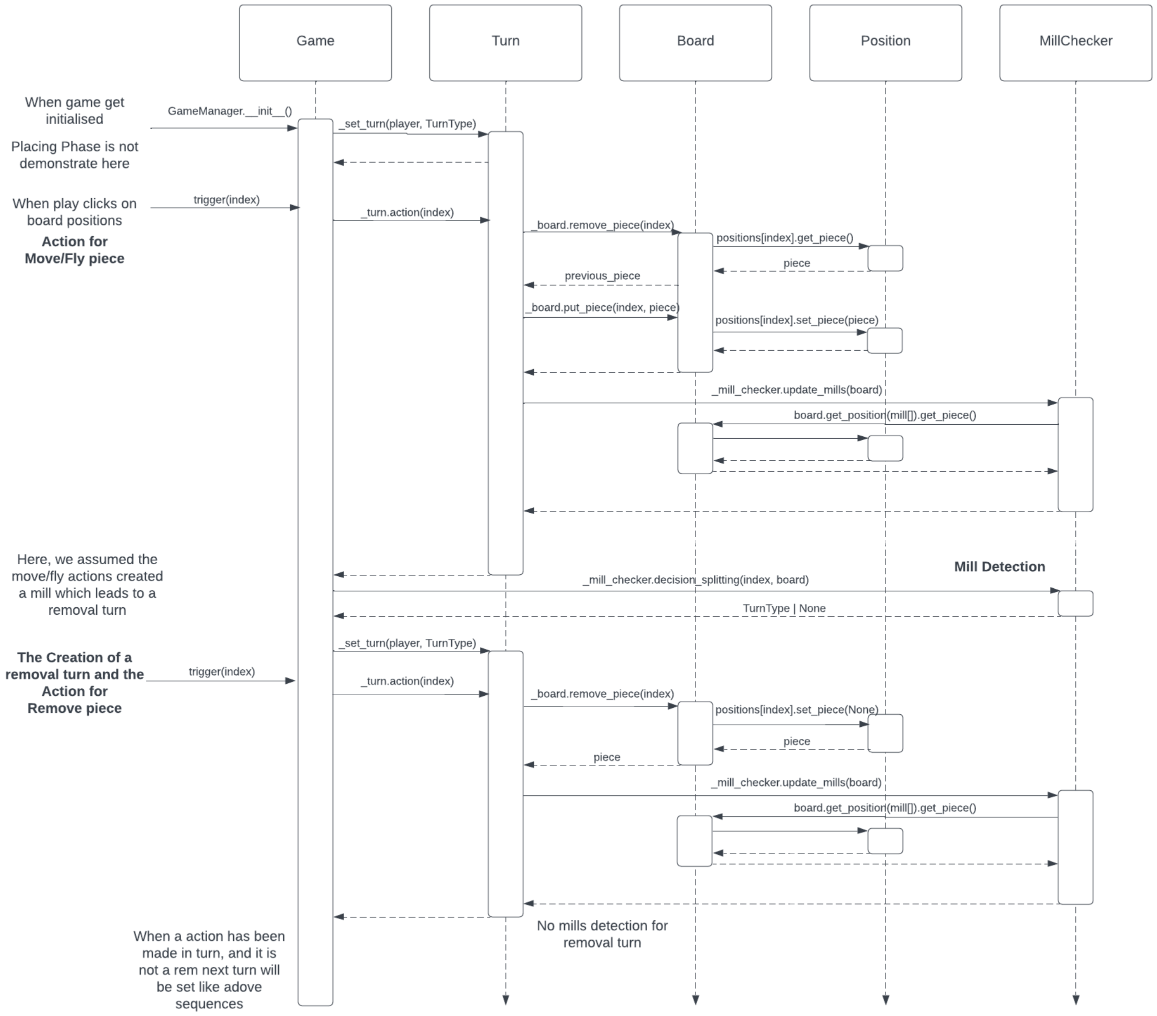
Demo Video

https://youtu.be/6tlHGIQ1RIo

Diagram Link:

https://lucid.app/lucidchart/0abd0599-3e18-4457-be1e-996aeab66dd3/edit?viewport_loc=-251%2C192%2C2219%2C1021%2C0_0&invitationId=inv_98352abf-b846-450b-9fe6-6812cb867b06

Sequence Diagrams | Sequence of Game, Turn initialised and interact with classes


Game
Turn
Board
Position
MillChecker
When game get initialised
GameManager.__init__()
_set_turn(player, TurnType)
Placing Phase is not demonstrate here
When play clicks on board positions
trigger(index)
_turn.action(index)
Action for Move/Fly piece
_board.remove_piece(index)
positions[index].get_piece()
piece
previous_piece
_board.put_piece(index, piece)
positions[index].set_piece(piece)
_mill_checker.update_mills(board)
board.get_position(mill[]).get_piece()
Here, we assumed the move/fly actions created a mill which leads to a removal turn
Mill Detection
_mill_checker.decision_splitting(index, board)
TurnType | None
_set_turn(player, TurnType)
The Creation of a removal turn and the Action for Remove piece
trigger(index)
_turn.action(index)
_board.remove_piece(index)
positions[index].set_piece(None)
piece
piece
_mill_checker.update_mills(board)
board.get_position(mill[]).get_piece()
No mills detection for removal turn
When a action has been made in turn, and it is not a rem next turn will be set like adove sequences

Design Rationales

○ Explain why you have revised the architecture, if you have revised it. What has changed should be covered in the previous point. This one is about why it changed.

- We added a Player class to better achieve responsibility separation. Previously, our player attributes composed only the player colour and had no real connection to other classes. However, our game end states came to rely on the access of the count of each player's number of remaining pieces. This gave too much responsibility to game.py to manage and was thus extracted. In addition, should the game need to host more than two players in the future, the extraction makes it easy to implement factory methods or applying interfaces for different types of players.

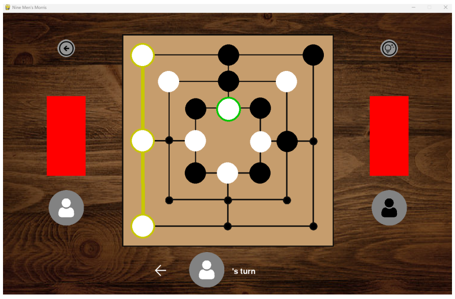
- Referring to the changes made to Player.py, Position.py also stores the piece based on the enum attribute stored in player. This reduced the redundancy of having two different objects store the same information that needed

- We removed the Piece class as it functioned solely as a data class.

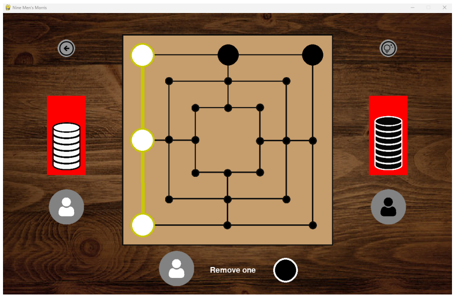
- Different turn types were extracted into different subclasses inheriting from a single base Turn class. The Move-type turn also extends from the Fly-turn as the underlying functionalities are the same. We did this so that the Trigger function could achieve Liskov Substitution, being able to use any variation of the Turn child classes without knowing the difference. This would also allow us to add new turn types in the future if necessary.

○ Explain 2-3 quality attributes (as non-functional requirements, e.g. usability, flexibility) that you consider relevant to the 9MM game and have explicitly considered in your design. Why are they relevant and important to your game? Show (provide evidence) how your design manifests these non-functional requirements.

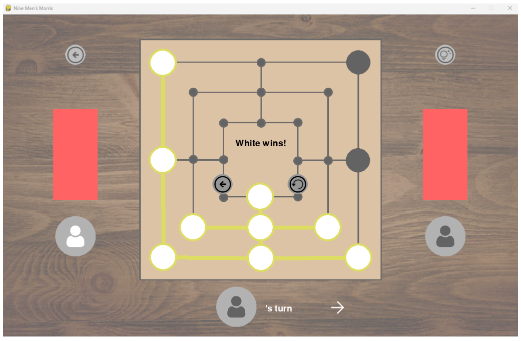
- **Reliability:**
  We have released a stable patch of the game, as of this time, there are no bugs or errors that we know of. We have rigorously tested all the requirements that were given to us (basic game rules) and the game runs flawlessly.



- **Usability:**
  When you click on your own pieces, it is highlighted green so that you know what piece you have selected, it also tells the players whose turn it is at the bottom and what type of turn/move they are to make.
- **Testability:**
  We made it so that our mills were highlighted in yellow. When we were testing, we moved a piece out of a mill and the background where the mill still existed in the system was still highlighted yellow. This let us know that our mill state was out of sync with the game and that we needed to fix our code.



○ Explain at least one human value (from Schwartz’s theory, e.g. achievement, tradition, freedom) that you consider relevant to the 9MM game and have explicitly considered in your design. Why is it relevant and important to your game? Show (provide evidence) how your design manifests this value.



Achievement:
When you win the game, there is a text as well as a cheer sound played informing the players who won. There is a sense of achievement in defeating an opponent in a game that is solved (meaning that the game should end in a draw if both sides play perfectly). Many games are played with the intent of having fun, winning and expressing skills and our 9 men's Morris is no different.

~ End ~