COMP3211: SOFTWARE ENGINEERING (2021-2022)

Group Project – Monopoly

API design document

Cheung Sui Wing (21027547D)
Lau Man Chun (21027257D)
Kwong Chun Him (21028468D)
Cheng Chi Kit (21028079D)

Table of Contents

1. Class/Method description

1.1. Main.py

The first py to be run, will import and call different py, such as call Menu.py for the player to choose “New Game” or “Continue”, and if the answer is “New Game”, Main.py will call the required py to start the game.

Variable		
Name	Type	Description
+block_list	List [Block]	To store the all block objects
+player_list	List [Player]	To store the all player objects

Method	
Name	main()
Argument	N/A
Return Type	void
Description	The main program, calls the print_menu() from menu.py, and get the return number and generate the players, blocks, and gameboard and run the game.

1.2. Data.py

Data.py stores all the data, in different forms, mainly in JSON, secondly is a enum class for storing colors.

Variable		
Name	Type	Description
+START_MONEY	INT	The amount of money present to the player when new game start
+SALARY	INT	The amount of money present to the player when the player passes the Start block
+MAX_TURN	INT	The maximum turns of the game

1.3. Menu.py

Menu.py is called by Main.py when the program starts, it’s use is to print the menu for the player to choose different options, which are “New Game”, “Continue”, and “Check Game Rule”.

Method	
Name	print_menu()
Argument	N/A
Return Type	INT
Description	Print the menu when the game starts. Ask the user to start a new game, continue the last game, check the game rule, or exit Return 0 for new game Return 1 continue game

1.4. GameBoard

GameBoard provides all the main logic of the program, first, print the game board with similarity same as the Monopoly game with all the details, houses, money, dice, etc. Second, it checks if the player is in jail, if not, call the roll dice method, after the roll dice method is finished and move the player to another block(square, box), it will call the activateBlockEffect method in Block.py to perform the expected action when landing on a block, such as when the player landed on a building, it will check if the building has been bought and if yes, the player will need to pay a rent, etc.GameBoard.py main method will keep looping before the maximum of turn have been reached or the player chooses to save the same and continue later.

Variable		
Name	Type	Description
+turn	INT	The current turn of this game
+players	List [Player]	The player object list
+blocks	List [Block]	The block object list
+jailList	List [Player]	A list of players who is in jail
+fine	INT	Fine needed to pay for getting out of the jail
+current_player	Player	The current player object

Method	
Name	__init__()
Argument	players: List [Player] blocks : List [Player]
Return Type	void
Description	Constructor of GameBoard

Method	
Name	print_board()
Argument	N/A
Return Type	void
Description	Print the game board on the command line

Method	
Name	set_currentPlayer()
Argument	player : Player
Return Type	void
Description	Set the current player

Method	
Name	roll_dice()
Argument	N/A
Return Type	Void
Description	Call the dice() and move the player to new position

Method	
Name	add_to_jail_list()
Argument	player: Player fine : int
Return Type	void
Description	Add the player to the jailList and set the fine number

Method	
Name	save_game()
Argument	N/A
Return Type	void
Description	Record all the game status and output a save file

Method	
Name	roll_dice_face()
Argument	N/A
Return Type	INT
Description	Random the index of the dice face then return (0-3)

Method	
Name	dice()
Argument	N/A
Return Type	INT
Description	Call roll_dice_face() then get the target face, then randomize the index (0-2) and get the final dice number, then return

Method	
Name	run()
Argument	N/A
Return Type	Void
Description	Run the gameboard with the logic in the functional requirement.

1.5. Block.py

Class Block stores all the logic to change any data when the player landed on a specific block, and the activateBlockEffect method, each block will perform different action and check if the player has the required data to perform the action, such as enough money to pay the rent.

Abstract Class: Block

Variable		
Name	Type	Description
+block_data	Dict	The block data
+position	List [Player]	The block position
+name	List [Block]	The block name

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of the Block

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	This method will execute the logic of the block, default, there is no effect, only provide a save game option.

Class: Start

Variable		
Name	Type	Description
+sub_text	Str	The subtext of the block

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of Start

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	No Effect, call super()

Class: Property

Variable		
Name	Type	Description
+price	INT	The price of the property
+rent	INT	The rent that the player needs to pay
+owner	Player	The property owner

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of Property

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	When the property has no owner, ask the player to buy it When the porterty has owner, ask the player to pay the rent

Method	
Name	reset_owner()
Argument	N/A
Return Type	Void
Description	Reset the owner to None

Method	
Name	set_owner()
Argument	player : Player
Return Type	Void
Description	Set the player to be the owner

Class: IncomeTax

This object used to handle if a player lands on this square, she pays 10% of her money (rounded down to a multiple of 10) as tax.

Variable		
Name	Type	Description
+subText	Str	The subtext of the block
+tax	INT	The tax %

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of Income Tax

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	Calculate the price that the player have to pay and subtract the money of the player

Class: Jail

Variable		
Name	Type	Description
+subText	Str	The subtext of the block
+fine	INT	The fine needed to pay for getting out of the jail
+turn	INT	The max turns that the player need to stay in the jail

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of Jail

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	No Effect, call super()

Class: Chance

Variable		
Name	Type	Description
+subText	Str	The subtext of the block
+min	INT	Minimum price player have to pay
+max	INT	Maximum price player can get

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of Chance

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	Randomize the number 0-1 to see if the player losees or gains money Then, calculate the amount of money of the player that need to be changed

Class: FreeParking

No effect of this block

Variable		
Name	Type	Description
+subText	Str	The subtext of the block

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of FreeParking

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	No Effect, call super()

Class: GoToJail

Variable		
Name	Type	Description
+jail_position	INT	The position of Jail Block
+fine	INT	The fine needed to pay for getting out of the jail
+turn	INT	The max turns that the player need to stay in the jail

Method	
Name	__init__ ()
Argument	block_data : Dict
Return Type	Void
Description	Constructor of GoToJail

Method	
Name	activate_block_effect()
Argument	player : Player game_Board : GameBoard
Return Type	Void
Description	Set the player position to the jail position Set how many turn(s) left before the player leaves the jail

1.6. Player.py

Player.py is the object that stores the properties, attributes of the players, such as the number of players, is the player alive or not, and is the player in jail or not, it will also perform the action to pay any money such as rent, and money when landed on the "Chance" block if unlucky, or jail, moreover, add money, such as when the player reached the "Start" block, other player landed on the player property(s), and when landed on the "Chance" block if lucky.

Variable		
Name	Type	Description
+player_number	INT	The player number (1-6)
+money	INT	The money that the player have
+position	INT	The position of the player in the game board
+jail_left	INT	How many turns that the player should stay at the jail

Method	
Name	__init__ ()
Argument	player_number : int money : int position : int
Return Type	Void
Description	Constructor of Player

Method	
Name	pay_money()
Argument	money : int
Return Type	Void
Description	Subtract the player's money according to the argument

Method	
Name	add_money()
Argument	money : int
Return Type	Void
Description	Add the player's money according to the argument

Method	
Name	is_alive()
Argument	N/A
Return Type	Boolean
Description	If the player's money >= 0, return True, If not, return False

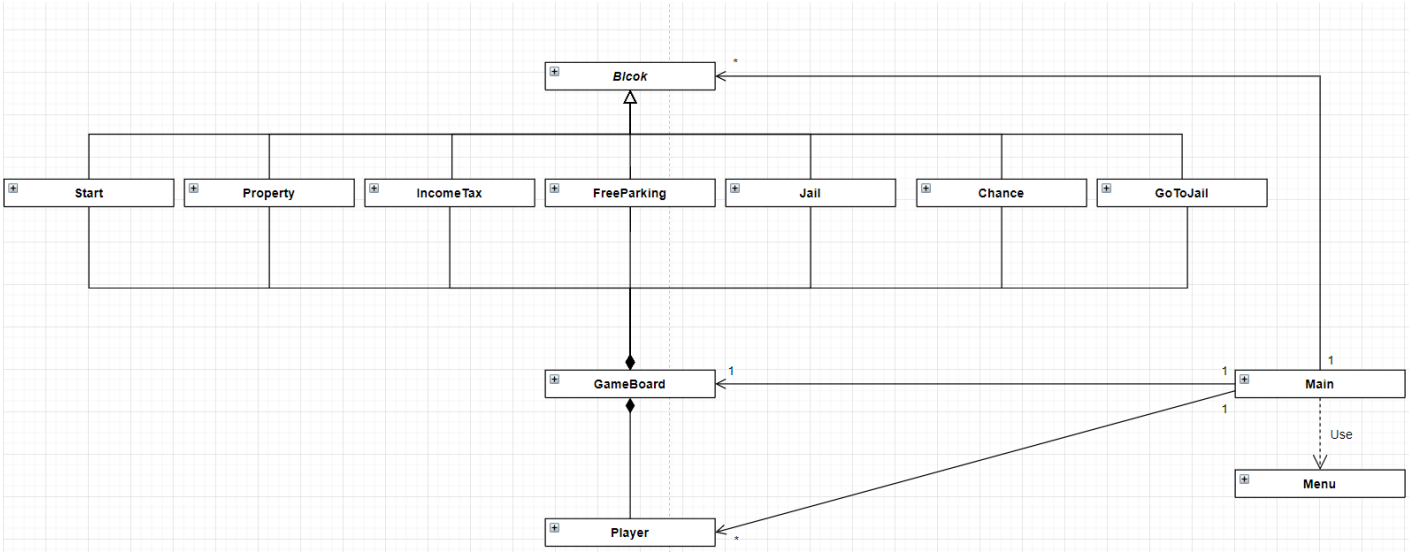
Method	
Name	is_in_jail()
Argument	N/A
Return Type	Boolean
Description	If the number of jail_left is 0, return False If not, return True

2. System models

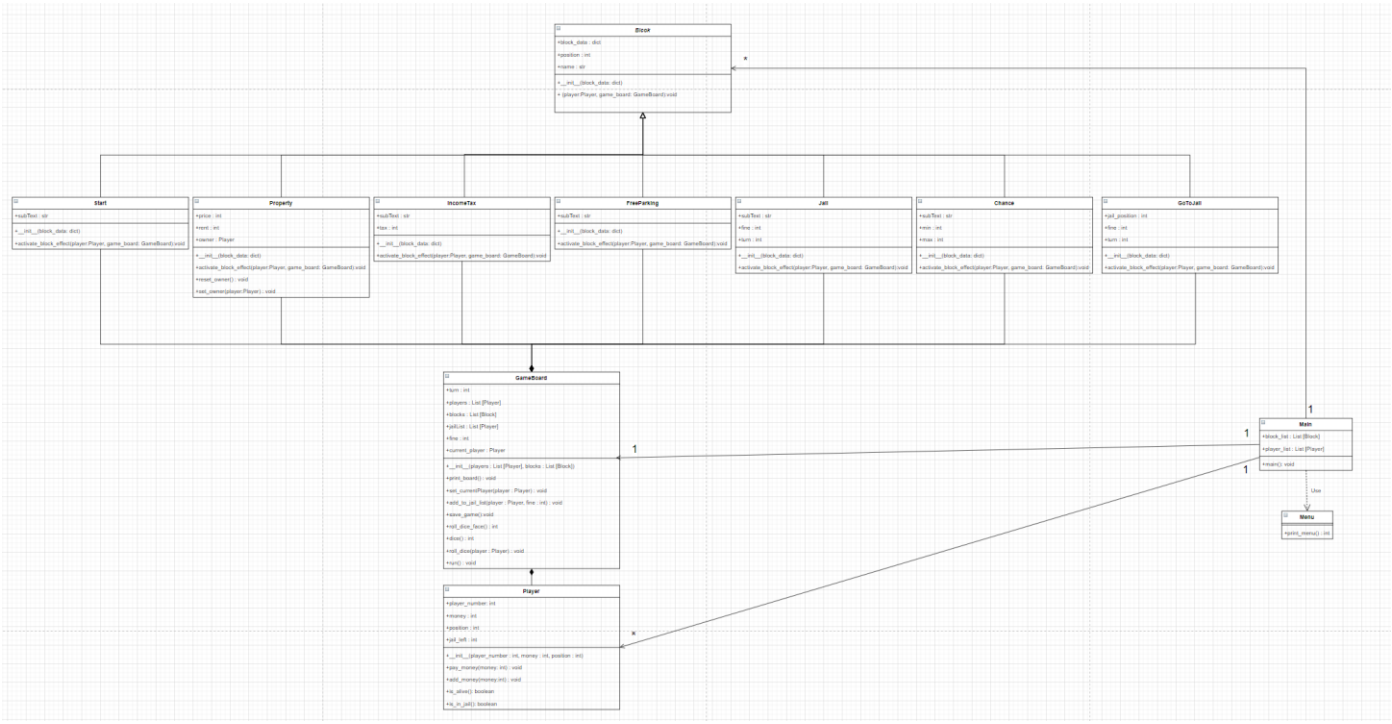
2.1. Class Diagram

2.1.1. Overall

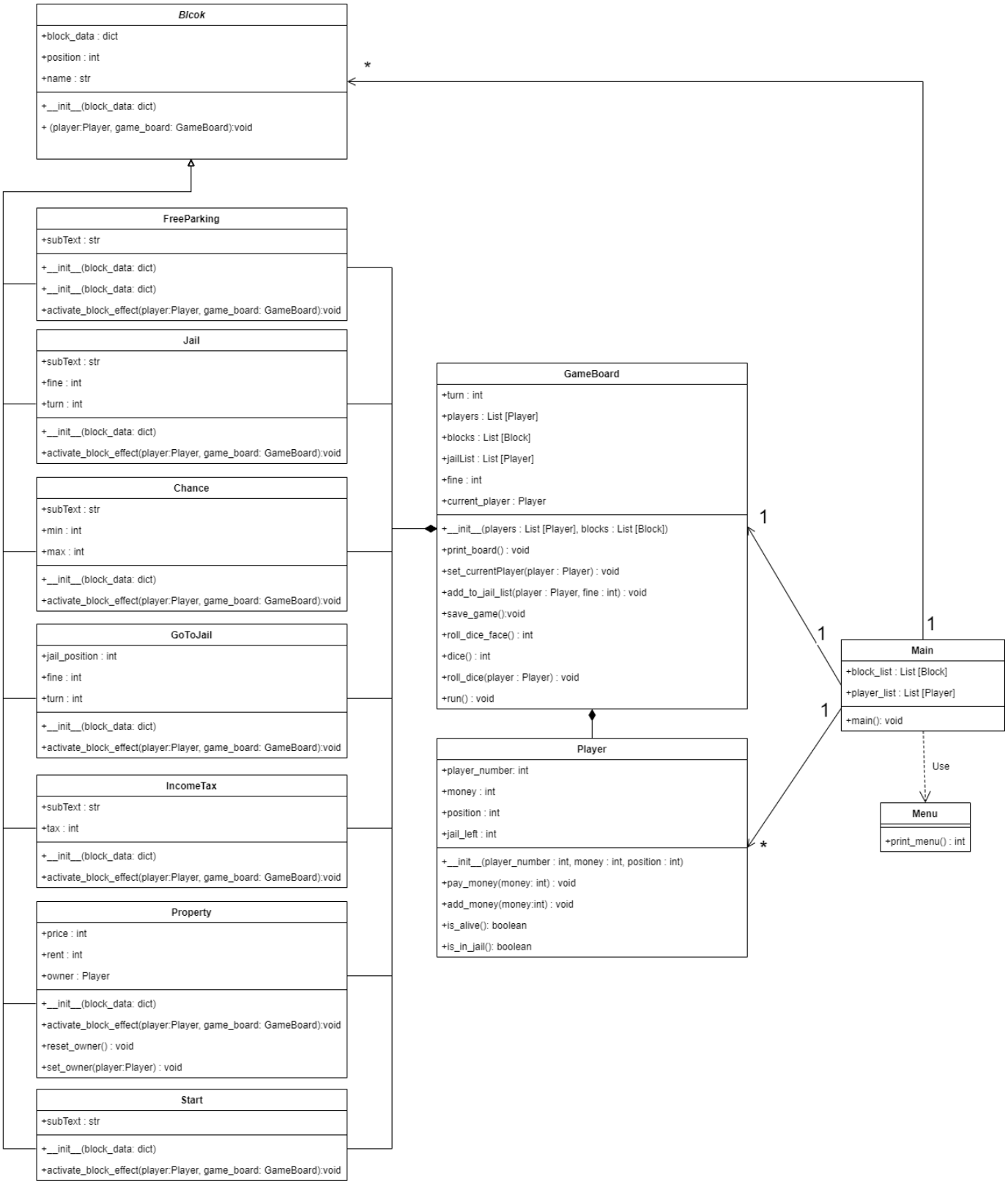
Simple version:



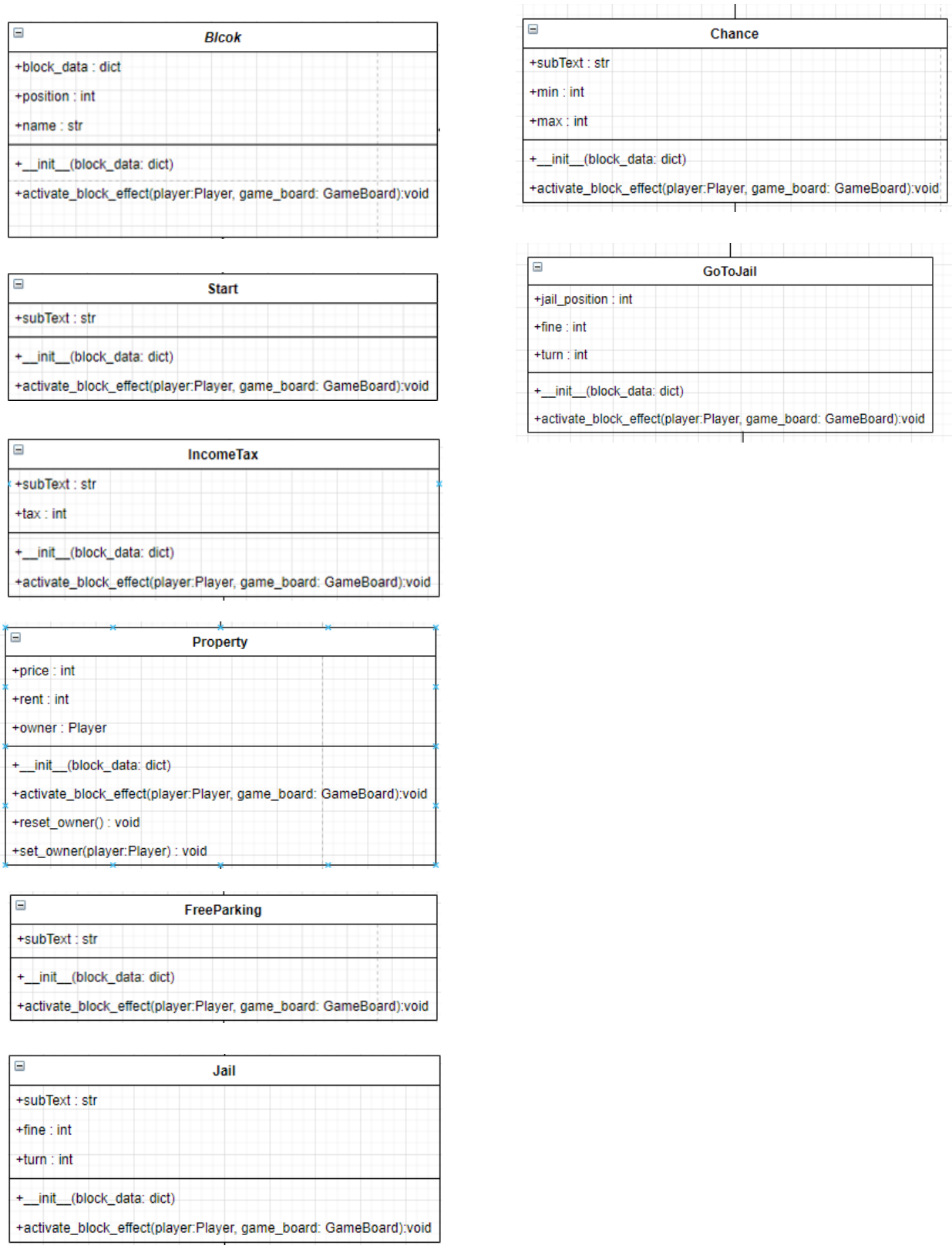
Detailed version (horizontal):



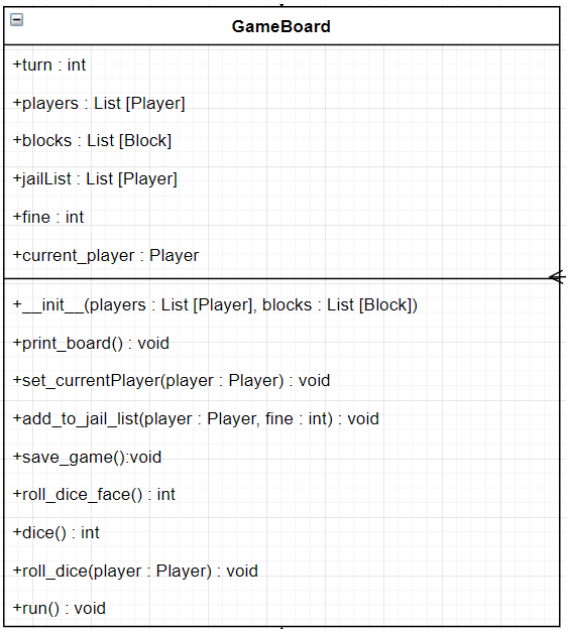
Detailed version (vertical):



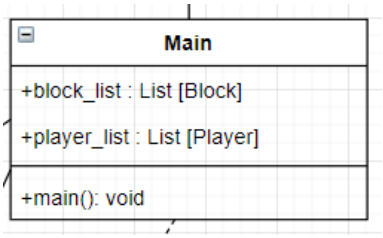
2.1.2. Block


Blcok
+block_data : dict
+position : int
+name : str
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
Chance
+subText : str
+min : int
+max : int
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
Start
+subText : str
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
GoToJail
+jail_position : int
+fine : int
+turn : int
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
IncomeTax
+subText : str
+tax : int
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
Property
+price : int
+rent : int
+owner : Player
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
+reset_owner() : void
+set_owner(player:Player) : void
FreeParking
+subText : str
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void
Jail
+subText : str
+fine : int
+turn : int
+__init__(block_data: dict)
+activate_block_effect(player:Player, game_board: GameBoard):void

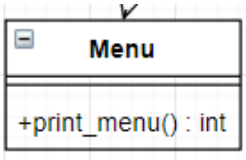
2.1.3. GameBoard



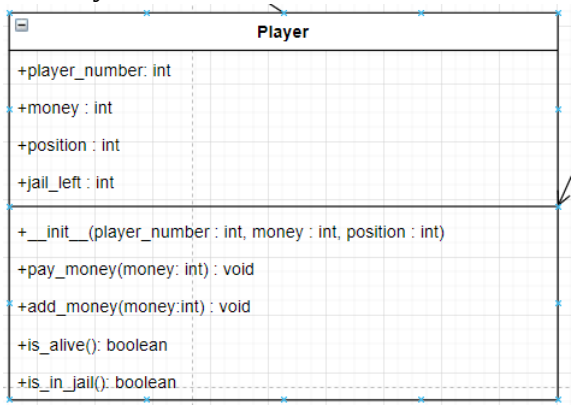
2.1.4. Main



2.1.5. Menu

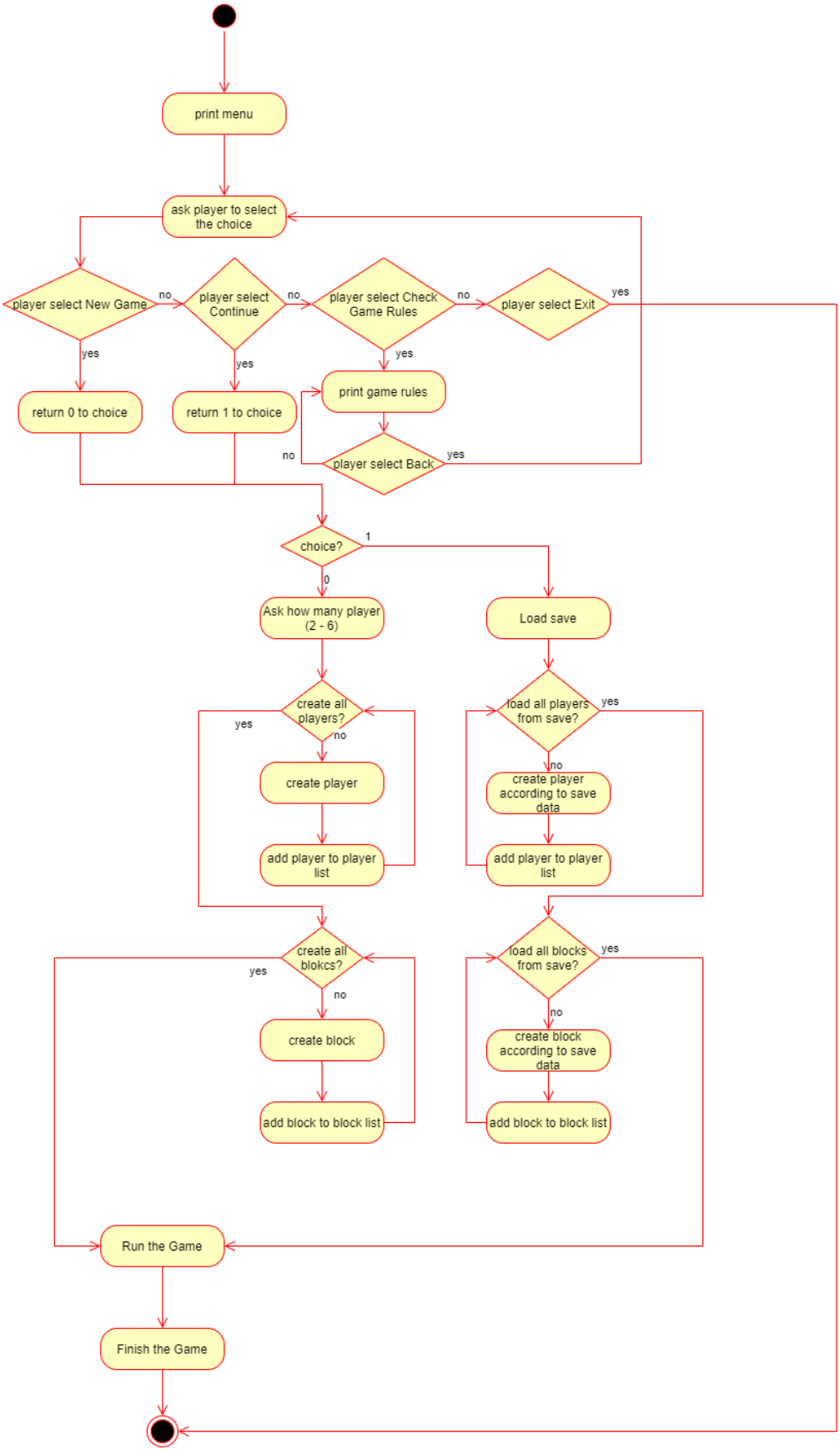


2.1.6. Player

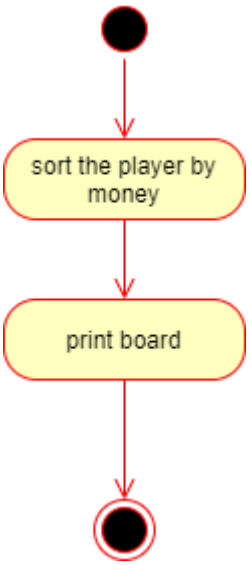


2.2. Activity Diagram

2.2.1. Main

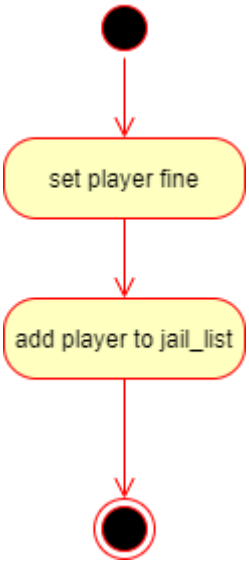


2.2.2. Print Board

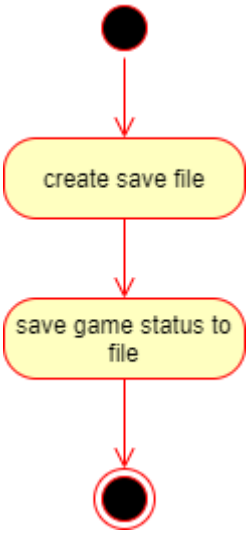


2.2.3. Roll dice

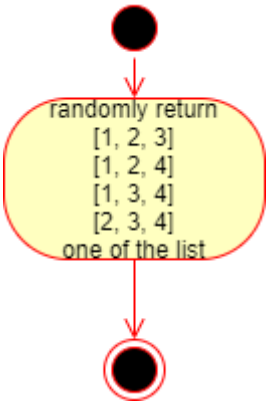
2.2.4. Add to jail list



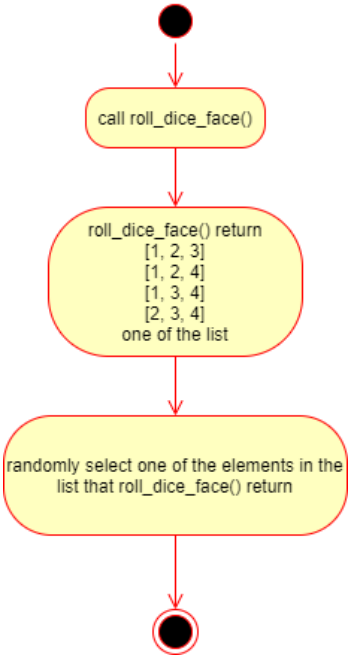
2.2.5. Save game



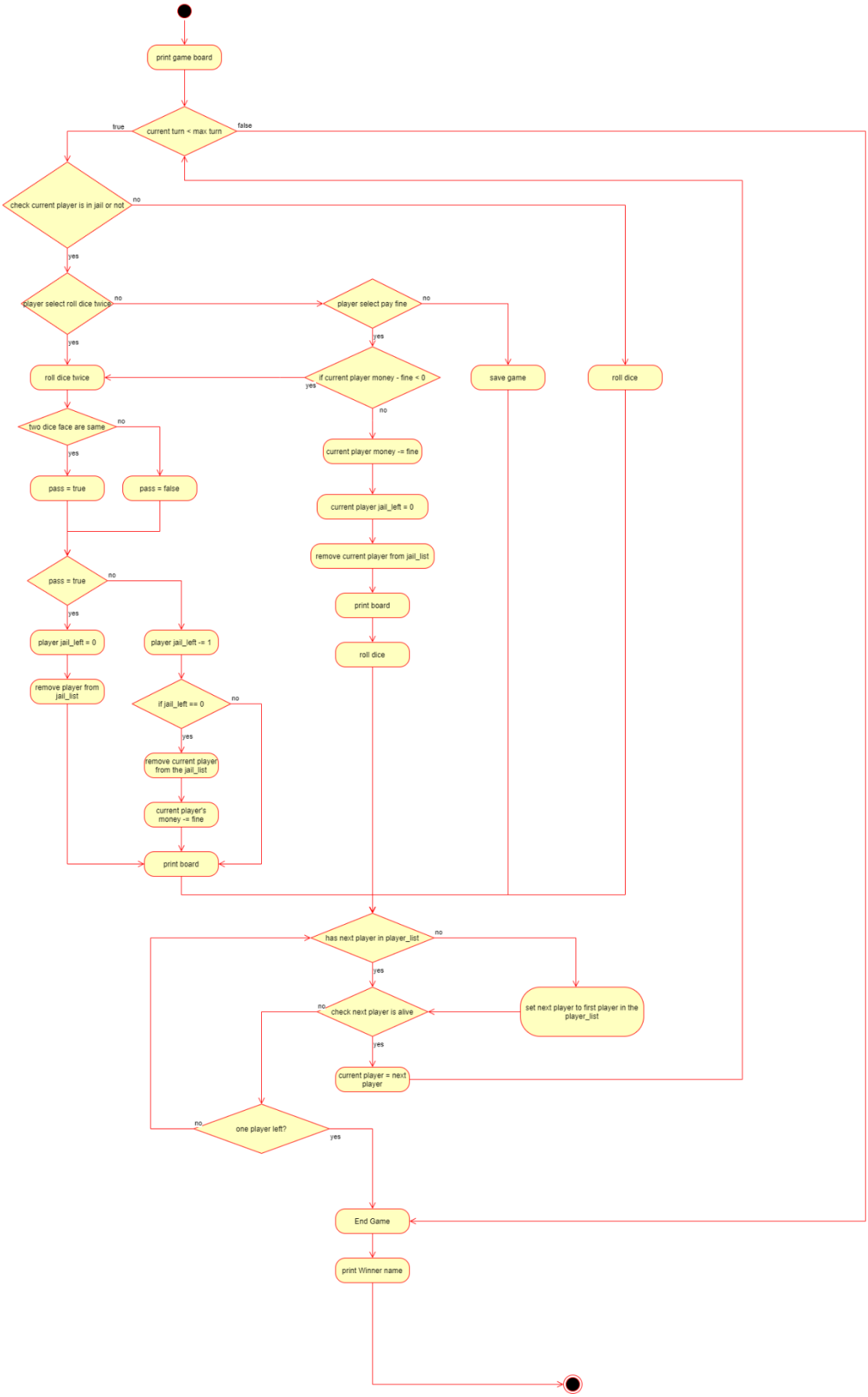
2.2.6. Roll dice face



2.2.7. Dice

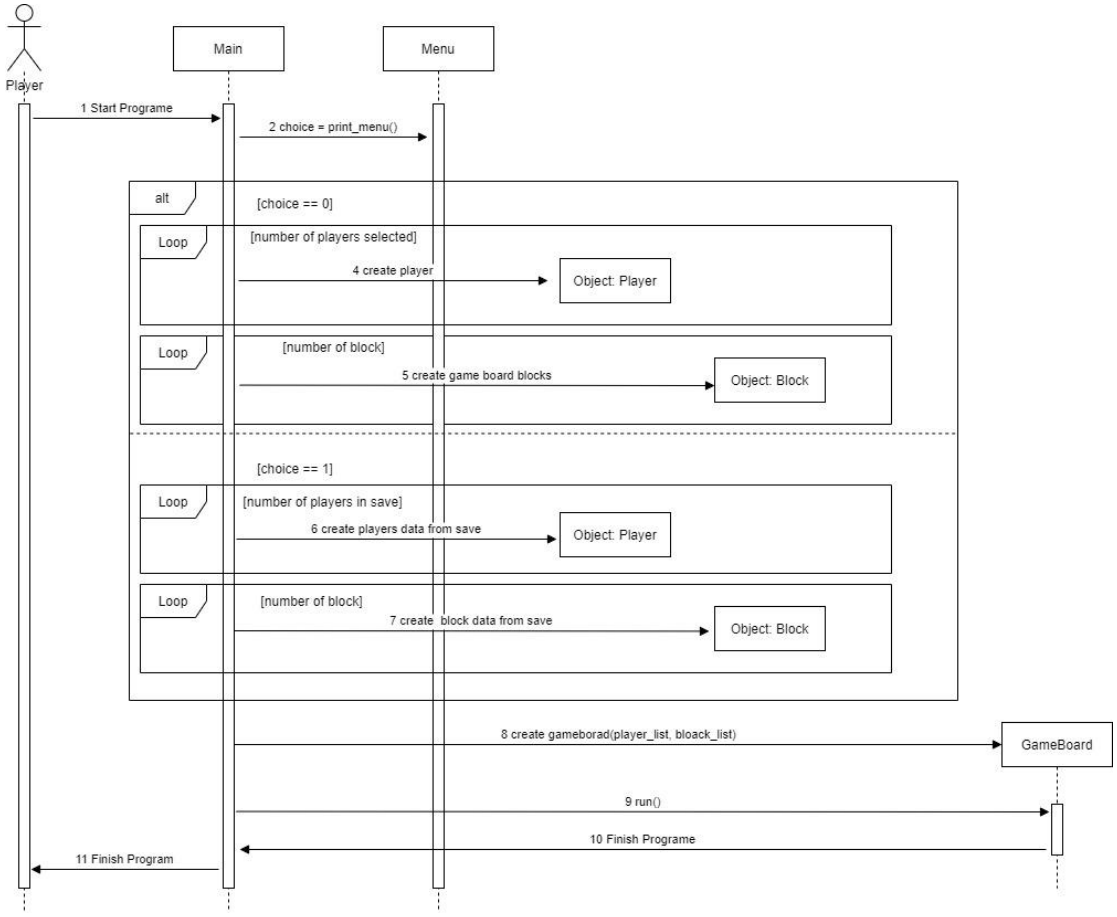

call roll_dice_face()
roll_dice_face() return
[1, 2, 3]
[1, 2, 4]
[1, 3, 4]
[2, 3, 4]
one of the list
randomly select one of the elements in the list that roll_dice_face() return

2.2.8. Run(main game logic)



2.3. Sequence Diagram

2.3.1. Main



2.3.2. run(Main Game Logic)