Design Overview for Way to World Cup

Name: TIEN DAT PHAM
Student ID: 103174539

Summary of Program

The SplashKit library is used to generate visuals and animations in the game, which is coded in C#. The game is named "Way To World Cup," and it was based on the Monopoly board game. The game may be played by two to four players, and it ends when three players lose (in this case, run out of money). In essence, the game's mechanism of action is similar to that of real-life Monopoly , including the ability for players to trade for estate, roll dice, be imprisoned, or own cards with magical abilities.

Each player will begin with $3000, and to move across the board, they will roll the dice. The asset cell is the most common sort of cell on the game; depending on the player's choice, they can be purchased or not. The main function of the game is that players have the right to buy real estate in cities around the world, and when other players set foot in cities that are already owned by someone, they will have to pay rent. When there is only 1 player left on the board, the game will stop.

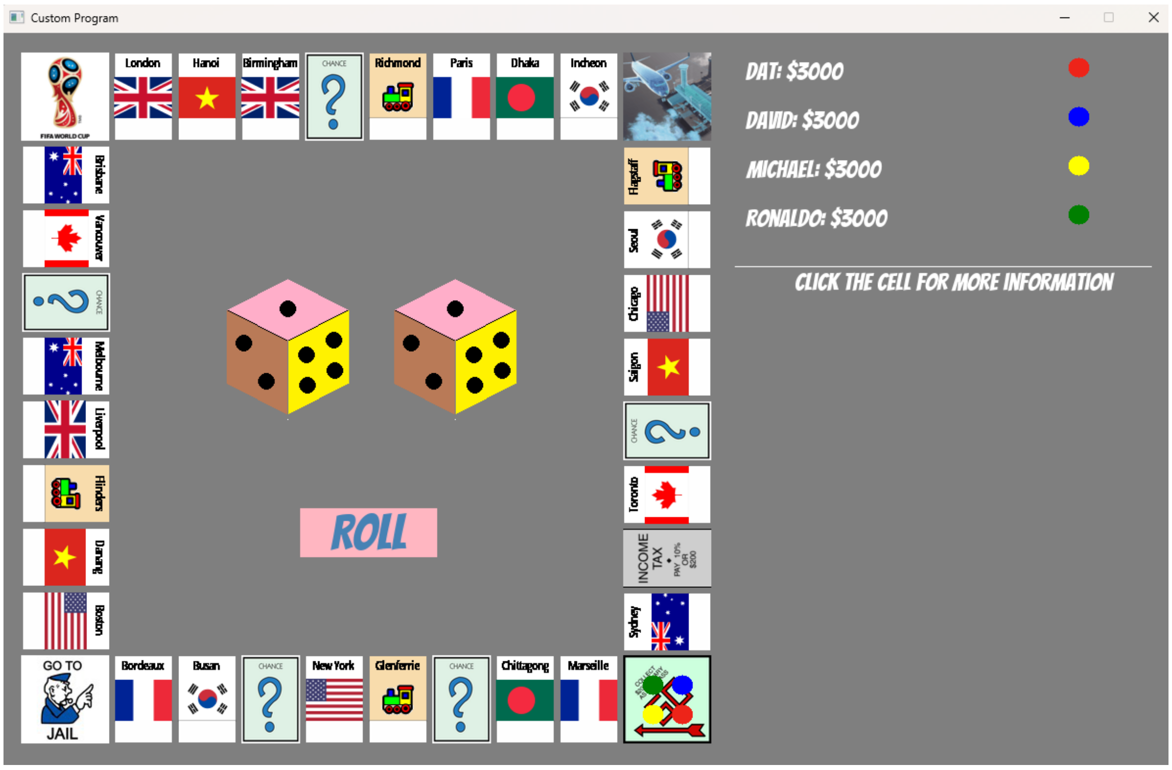


Figure 1 : Interface of the game

Required Roles

Interfaces

Table 1: MouseClickedEvent

Responsibility	Type Details	Notes
knows mouse in the right position or not	IsAt(Point2D point): bool	using SplashKit.Point2D

Classes

Table 1: AffordableCell

Responsibility	Type Details	Notes
Knows to cell owned by who	_belongTo : Player	
Know the renting cost which already bought by another player	_rentCost : int[]	
Knows the cost to pay	_cost : int	
Get position of the cell on the board	AffordableCell(float x, float y, string name, Bitmap Image)	
Get the value of the city	Value : int <<virtual>>	
Knows is the player step on that cell or not	OnCellFunction (Player player) <<overrride>>	

Table 2: Airport

Responsibility	Type Details	Notes
Get airport cell position	Airport(float x, float y, string name, Bitmap image, Board board)	
Know the player has landed on airport or not	OnCellFunction (Player player) << override >>	

Table 3: Board

Responsibility	Type Details	Notes
Knows every cells	_cells: List<Cell>	
knows which class create the cell	_cellFactory: CellFactory	based on Factory pattern

Add more cell	AddCell(Cell c)	
Finding cell based on the position	FindCell(int coordinate): Cell	
Loading initialization data	Load(string filename)	
Get the how many cells on the board	CellNumber: int	

Table 4: Button

Responsibility	**Type Details**	**Notes**
Information of the button	Button (float x, float y, int width, int height, string name, Color color, Color hoveringColor, Color textColor)	
Check if the mouse on the button	IsAt(Point 2D point) : bool <<virtual>>	
Know when the mouse is clicked	OnClick (EventArgs e)	
When the mouse click	ClickEvent : EventHandler << event>>	
Button's name	Name : string <<property, readonly>>	

Table 5: Card

Responsibility	**Type Details**	**Notes**
Know which type of card	_description : string	
Activate the card	Activate (Player player, Board board) << abstract >>	

Table 6: Cell

Responsibility	**Type Details**	**Notes**
Get the function of the card when player step on that	OnCellFunction(Player player)	
Knows its name	_name, Name: string	
Loading initialization data	Load(StreamReader reader)	using System.IO.StreamReader
Assign image for the cell	Image : Bitmap << property, readonly >>	
Coordinate of the cell	Coordinate : int <<property>>	

Table 7: CellFactory

Responsibility	**Type Details**	**Notes**

Get which type of cell going to be created	Dictionary<string, Type>	
Get which type of image going to assigned for the cell	Dictionary<string, Bitmap>	
Register the cell type	RegisterCell(string typeName, Type type)	
Register the images	RegisterImage(string typeName, string filename)	
Create a cell based on its type	CreateCell(string typeName, float x, float y, string name, Board board)	based on Factory pattern

Table 8: Database

Responsibility	**Type Details**	**Notes**
Get database information	_database: Database	using SplashKit.Database
Knows the instance	_instance: GameDatabase	GameDatabase is a singleton
Gets the instance	GetDatabase(): GameDatabase	
Using query to handle the database	Query(string sql): QueryResult	using SplashKit.QueryResult
Free the database and all submitted queries	FreeDB()	
Check to see the database includes cell's information or not	DataExit()	
Intialize database information	InitCellData(string name)	

Table 9: Dice

Responsibility	**Type Details**	**Notes**
Know the value	Value: int	
Rolling the dice	Roll()	
Reset the dice	Reset()	
Disable the dice	Deactivated()	
Check if the dice stop rolling or not	EndRolling() : bool	

Table 10: DrawableObject

Responsibility	**Type Details**	**Notes**
Get x coordinate	_x : float	
Get y coordinate	_y : float	

Table 11: FileExtensionMethods (static class)

Responsibility	Type Details	Notes
Use StreamReader to read the integer datatype	ReadInteger(this StreamReader reader): int	
Use StreamReader to read the float datatype	ReadFloat(this StreamReader reader): float	

Table 12: Game Implementation

Responsibility	Type Details	Notes
Knows the players	_players: List<String> players	
Knows the board	_board: Board	
Knows the dices	_dice1: Dice, _dice2: Dice	
Knows the buttons	_buttons: List<Button>	
Knows the notification box	_notiBox: GameNotifications	
Knows the sidebox notification	_sideNotiBox: GameNotifications	
Knows the side bar image	_sideBarImage : Bitmap	
Knows the turn	_turn : int	
Check how many players left	PlayersLeft : List<Player>	
Updates the game	Update()	Control the game

Table 23: Game Notifications

Responsibility	Type Details	Notes
Notifying for the player	_noti : string	

Table 34: GamingTools (static class)

Responsibility	Type Details	Notes
Can pause the screen so player can read the message of the game	DisplayDelay(uint time, Action action)	
Get a new position when rotating a cell around center point	FindRotatePoint(float centerX, float centerY, float x, float y, float angle) : Point 2D	Using SplashKit

Table 45: Jail

Responsibility	Type Details	Notes
Cell's responsibilities		

Table 56: MoneyCard (Card's responsibilities)

Responsibility	Type Details	Notes
Activate the function of the card	Activate (Player player, Board board)	Get or take money from player

Table 67: MouseInputManager

Responsibility	Type Details	Notes
Know the number of user have clicked	_observers : List<MouseClickedEvent>	Based on the Observer Pattern

Table 78: MoveCard (Card's responsibilities)

Responsibility	Type Details	Notes
Activate the function of the card	Activate (Player player, Board board)	Move player position backward/ forward

Table 89: Mystery

Responsibility	Type Details	Notes
Knows which type of mystery card	_cards : Dictionary <Type, Card> <<static>>	
Pick the card (money or move card)	_chooseCard : Card	2 kinds of mystery card : Move Card and Money Card

Table 20: Player

Responsibility	Type Details	Notes
Know and set the money	_money, Money: int	
Set position for player	_position, Position: int	
Set player's name	_name: string	
Can move player's position	MoveTo(float x, float y)	
Buy a property	Purchase(AffordableCell c)	
Build a house/resort on that land	Build(Property c, int cost)	
Sell all properties	SellAll()	
Ending the game	DeactivatedAllActions()	
Rolling the dice	RollPlan()	
Knows the total money of each player	TotalMoney : int <<property, readonly>>	

Table 29: PlayerGenerator

Responsibility	Type Details	Notes

Knows how many player going to play the game	GetPlayerCount()	
Knows the name of the player	GetPlayer(int playerCount)	

Table 22: Property

Responsibility	Type Details	Notes
Knows maximum house player can build in each cell	MaxHouse : int <<const >>	3 houses maximum
Get the type of property	_type : int	
Register type of property for each image	RegisterImg(int type, string filename) << static >>	
Loading type of property	Load(QueryResult qr)	
Draw the property	Draw()	

Table 23: Start

Responsibility	Type Details	Notes
Cell's responsibilities		

Table 24: Tax

Responsibility	Type Details	Notes
Cell's responsibilities		

Table 25: TemporaryButton

Responsibility	Type Details	Notes
Button's responsibilities		

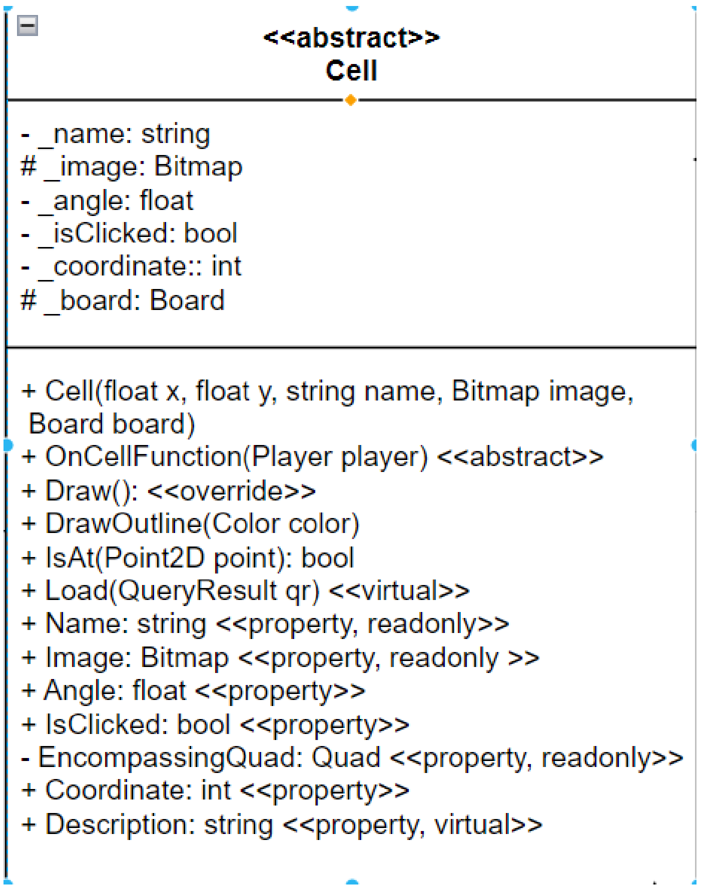
Table 26: TrainStation

Responsibility	Type Details	Notes
Know how max trainstation	MaxTrainStation : int <<const >>	

Table 27: WorldCup

Responsibility	Type Details	Notes
Cell's responsibilities		

Abstraction



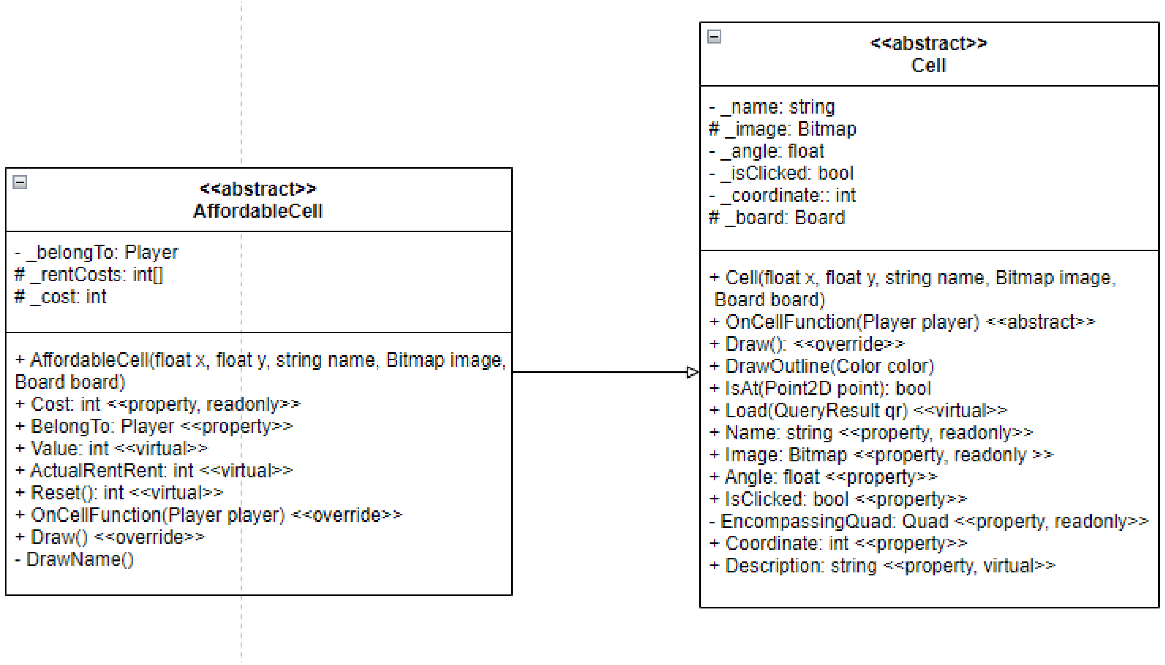
```
<<abstract>>
Cell

- _name: string
# _image: Bitmap
- _angle: float
- _isClicked: bool
- _coordinate:: int
# _board: Board

+ Cell(float x, float y, string name, Bitmap image,
 Board board)
+ OnCellFunction(Player player) <<abstract>>
+ Draw(): <<override>>
+ DrawOutline(Color color)
+ IsAt(Point2D point): bool
+ Load(QueryResult qr) <<virtual>>
+ Name: string <<property, readonly>>
+ Image: Bitmap <<property, readonly >>
+ Angle: float <<property>>
+ IsClicked: bool <<property>>
- EncompassingQuad: Quad <<property, readonly>>
+ Coordinate: int <<property>>
+ Description: string <<property, virtual>>
```

A great example of how to define an abstraction is my class Cell. Users may know about the Cell's name, position, what images are on it, and other fundamental details. Users may manipulate cells and move them across the board without having to know what unique purpose they will provide for the user. Basically, the user will have an overview of this Cell but the complex functions and implementation will be hidden.

Polymorphism

The Cell example above can also be used with the definition of Polymorphism. Cell, they basically have a lot of different forms. They can have many different functions such as owning the player's land, taking the player's money or maybe the starting Cell. Because Cell has so many characteristics, this is the Polymorphism I applied to my program.

Inheritance


<<abstract>>
AffordableCell
- _belongTo: Player
# _rentCosts: int[]
# _cost: int
+ AffordableCell(float x, float y, string name, Bitmap image, Board board)
+ Cost: int <<property, readonly>>
+ BelongTo: Player <<property>>
+ Value: int <<virtual>>
+ ActualRentRent: int <<virtual>>
+ Reset(): int <<virtual>>
+ OnCellFunction(Player player) <<override>>
+ Draw() <<override>>
- DrawName()
<<abstract>>
Cell
- _name: string
# _image: Bitmap
- _angle: float
- _isClicked: bool
- _coordinate:: int
# _board: Board
+ Cell(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<abstract>>
+ Draw(): <<override>>
+ DrawOutline(Color color)
+ IsAt(Point2D point): bool
+ Load(QueryResult qr) <<virtual>>
+ Name: string <<property, readonly>>
+ Image: Bitmap <<property, readonly >>
+ Angle: float <<property>>
+ IsClicked: bool <<property>>
- EncompassingQuad: Quad <<property, readonly>>
+ Coordinate: int <<property>>
+ Description: string <<property, virtual>>

An example of inheritance in my custom program is the relationship between two classes (AffordableCell and Cell). In general, AffordableCell is inherited from Cell and the features of AffordableCell are based on Cell. However, AffordableCell will have more obvious features than Cell like they will let users know if they have enough money to buy this Cell or not.

Class Diagram

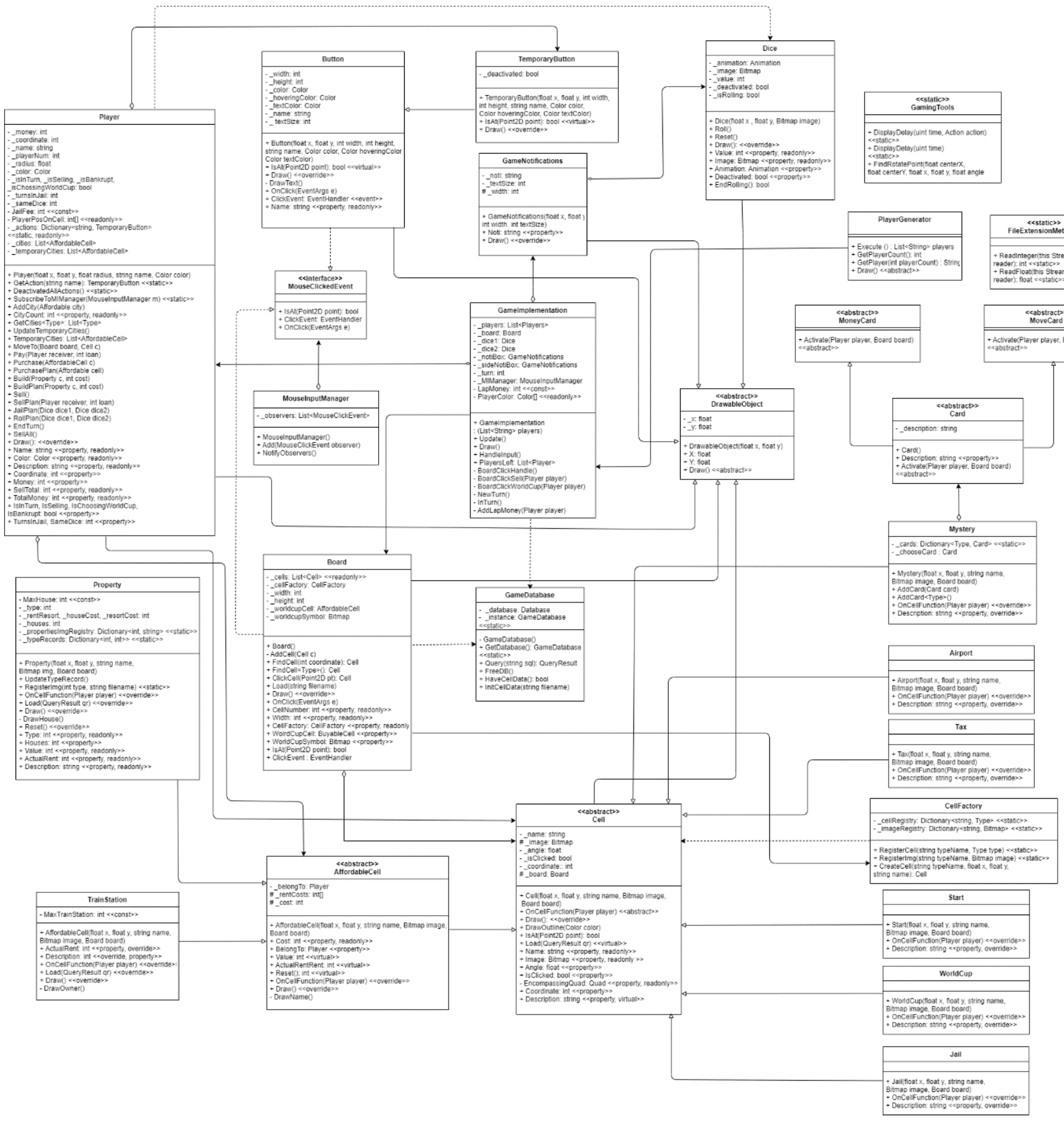


Figure 2 UML Diagram of Way To World Cup

Sequence Diagram

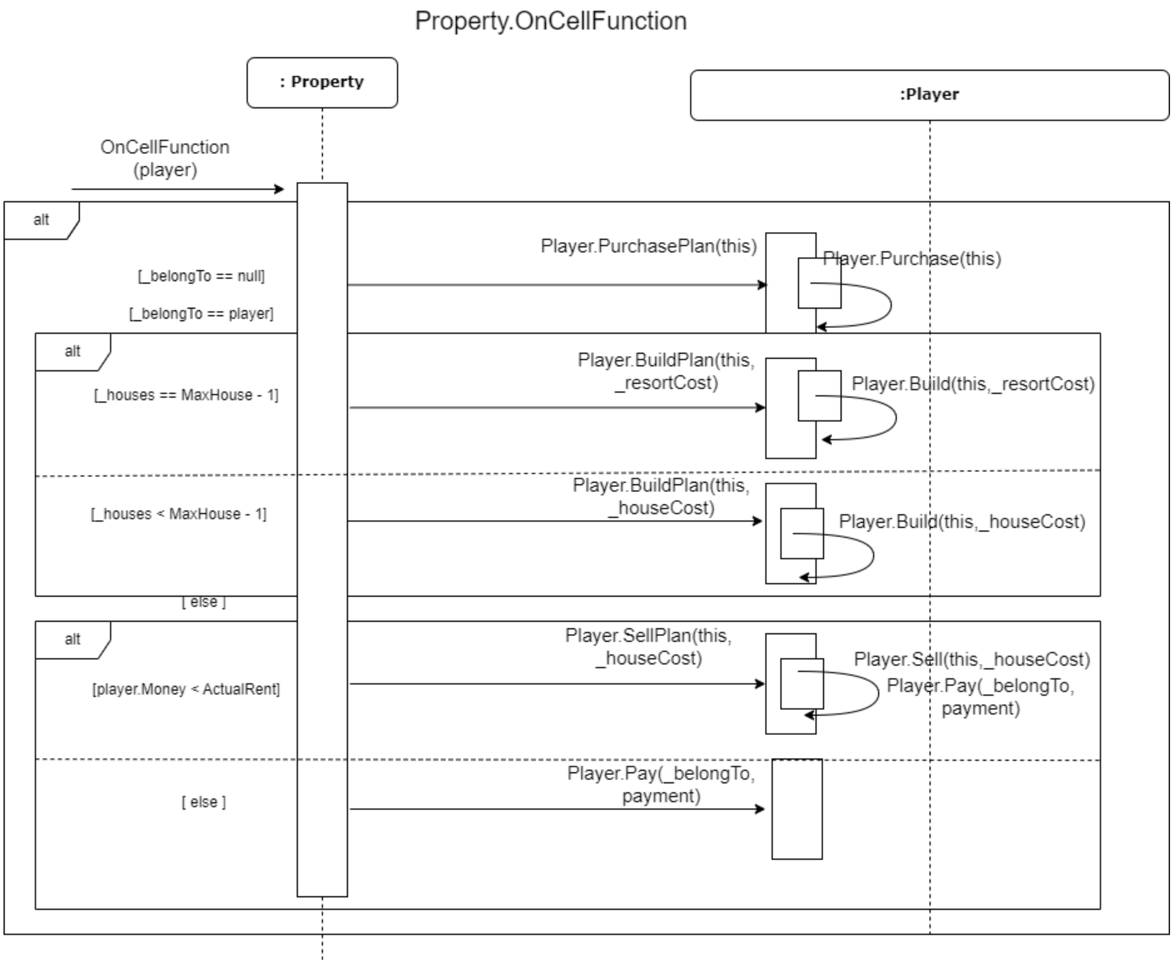

Property.OnCellFunction
: Property
:Player
OnCellFunction (player)
alt
[_belongTo == null]
[_belongTo == player]
Player.PurchasePlan(this)
Player.Purchase(this)
alt
[_houses == MaxHouse - 1]
Player.BuildPlan(this, _resortCost)
Player.Build(this,_resortCost)
[_houses < MaxHouse - 1]
Player.BuildPlan(this, _houseCost)
Player.Build(this,_houseCost)
[else]
alt
[player.Money < ActualRent]
Player.SellPlan(this, _houseCost)
Player.Sell(this,_houseCost)
Player.Pay(_belongTo, payment)
[else]
Player.Pay(_belongTo, payment)

Figure 3 Sequence Diagram of Way To World Cup

Design Patterns

1. Singleton

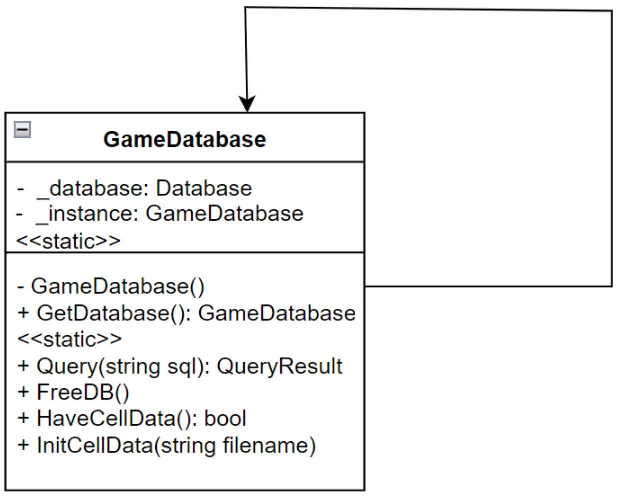

GameDatabase
- _database: Database
- _instance: GameDatabase <<static>>
- GameDatabase()
+ GetDatabase(): GameDatabase <<static>>
+ Query(string sql): QueryResult
+ FreeDB()
+ HaveCellData(): bool
+ InitCellData(string filename)

Figure 4 Example of Singleton Pattern

The Database class employs the Singleton Pattern. Users can call such functions without first making a class instance. It has one static method (GetDatabase), as shown in the UML diagram, and one static field (_instance). Despite defining it, _instance is an instance of the Singleton class, which is the class. The class so contains an instance of itself, which is declared static so that it does not need to create an instance of the Singleton class in order to access it. A good technique to provide a universal point of access to the instance is by using the Singleton pattern. When users need to establish a global object to reach from each activity of the system, this technique is quite helpful. The Singleton Database must be used since every class in my programme need a database to store data like properties, cell IDs, city names, etc.

2. Factory

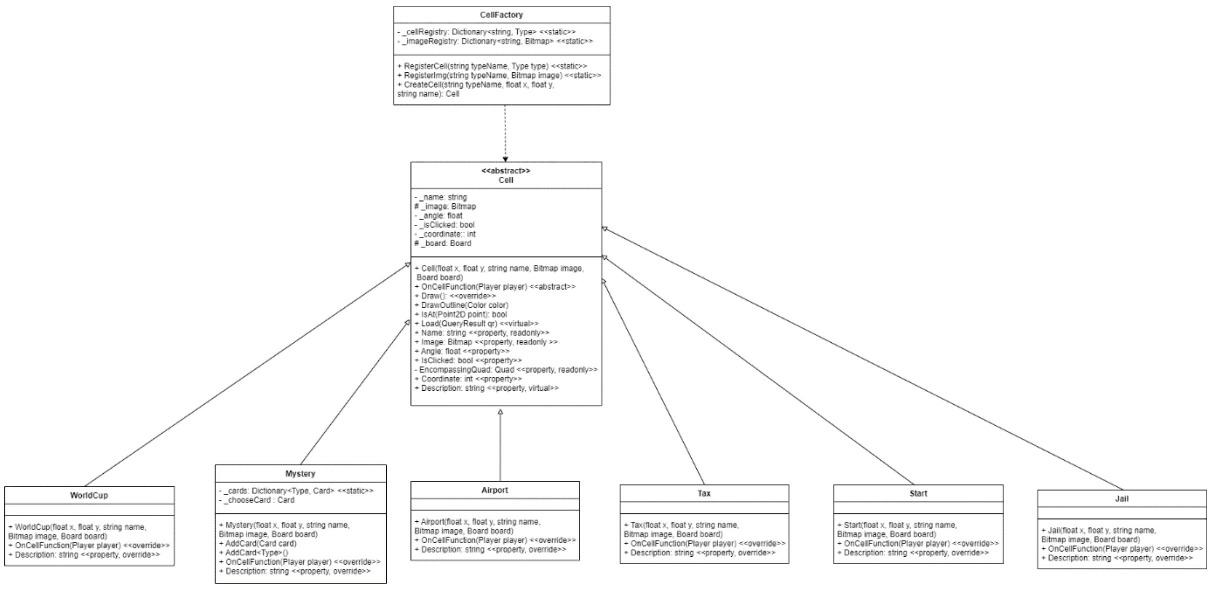

CellFactory
- _cellRegistry: Dictionary<string, Type> <<static>>
- _imageRegistry: Dictionary<string, Bitmap> <<static>>
+ RegisterCell(string typeName, Type type) <<static>>
+ RegisterImg(string typeName, Bitmap image) <<static>>
+ CreateCell(string typeName, float x, float y, string name): Cell
<<abstract>> Cell
- _name: string
# _image: Bitmap
- _angle: float
- _isClicked: bool
- _coordinate:: int
# _board: Board
+ Cell(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<abstract>>
+ Draw(): <<override>>
+ DrawOutline(Color color)
+ IsAt(Point2D point): bool
+ Load(QueryResult qr) <<virtual>>
+ Name: string <<property, readonly>>
+ Image: Bitmap <<property, readonly >>
+ Angle: float <<property>>
+ IsClicked: bool <<property>>
- EncompassingQuad: Quad <<property, readonly>>
+ Coordinate: int <<property>>
+ Description: string <<property, virtual>>
WorldCup
+ WorldCup(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>
Mystery
- _cards: Dictionary<Type, Card> <<static>>
- _chooseCard : Card
+ Mystery(float x, float y, string name, Bitmap image, Board board)
+ AddCard(Card card)
+ AddCard<Type>()
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>
Airport
+ Airport(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>
Tax
+ Tax(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>
Start
+ Start(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>
Jail
+ Jail(float x, float y, string name, Bitmap image, Board board)
+ OnCellFunction(Player player) <<override>>
+ Description: string <<property, override>>

Figure 5 Example of Factory Pattern

The factory method pattern advises switching out calls to the new operator for direct object formation calls with calls to a unique factory method. The Cell class, which specifies methods like OnCellFunction() and Description, should be implemented by classes like WorldCup, Mystery, Airport, Tax, Start, and Jail (). Each class implements this method in a unique way. For example, OnCellFunction() has a function that can assist the player in organising the WorldCup in other Cells when the player steps on the WorldCup tile, while OnCellFunction() has a function that allows players to play cards of any function when the player steps on the Mystery tile. The Cell class's attributes are essentially shared by these subclasses; the Cell class will only override the methods that were implemented in the subclasses.
This design pattern is excellent for reducing code duplication. To produce every cell without the factory pattern, we would need to contact 6 more operators. I currently just have a loop, though. New cell functions can also be easily added thanks to it.

3. Observers

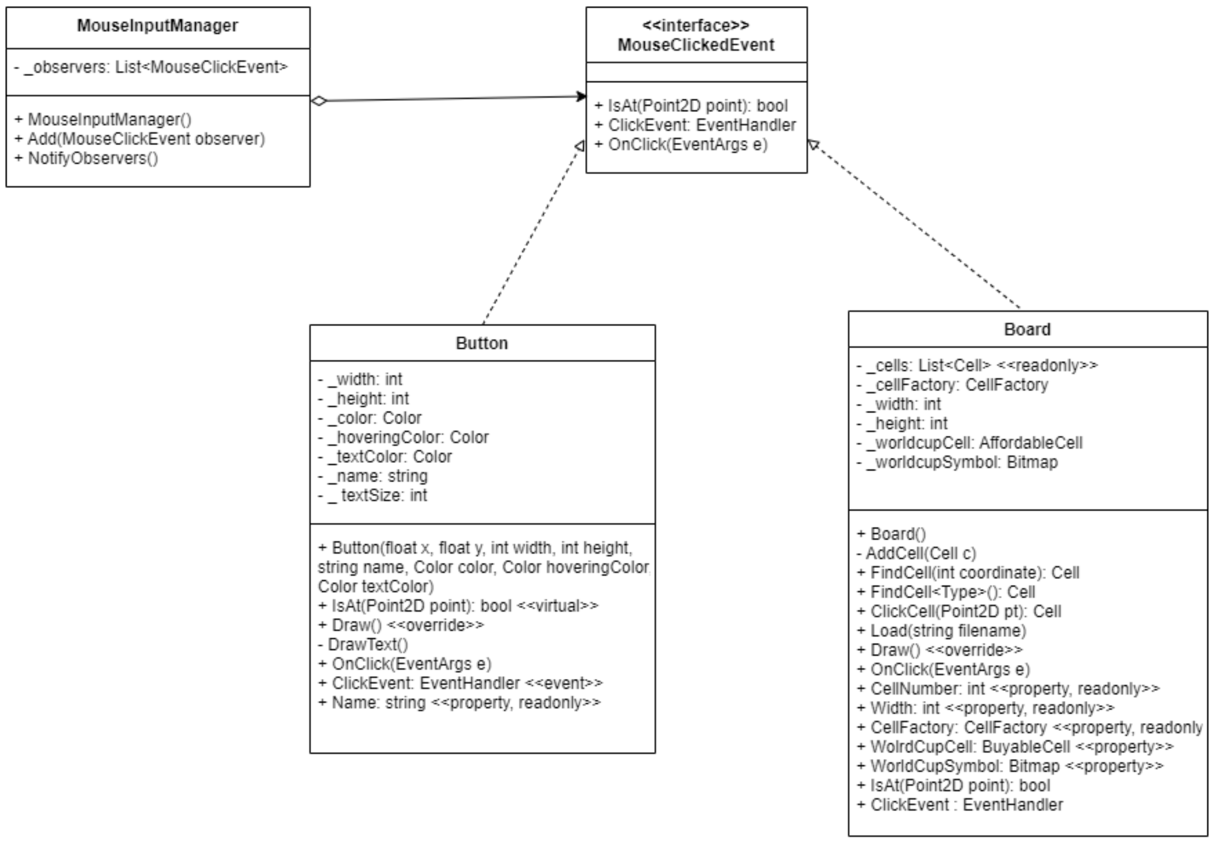

MouseInputManager
- _observers: List<MouseClickEvent>
+ MouseInputManager()
+ Add(MouseClickEvent observer)
+ NotifyObservers()
<<interface>>
MouseClickedEvent
+ IsAt(Point2D point): bool
+ ClickEvent: EventHandler
+ OnClick(EventArgs e)
Button
- _width: int
- _height: int
- _color: Color
- _hoveringColor: Color
- _textColor: Color
- _name: string
- _ textSize: int
+ Button(float x, float y, int width, int height, string name, Color color, Color hoveringColor Color textColor)
+ IsAt(Point2D point): bool <<virtual>>
+ Draw() <<override>>
- DrawText()
+ OnClick(EventArgs e)
+ ClickEvent: EventHandler <<event>>
+ Name: string <<property, readonly>>
Board
- _cells: List<Cell> <<readonly>>
- _cellFactory: CellFactory
- _width: int
- _height: int
- _worldcupCell: AffordableCell
- _worldcupSymbol: Bitmap
+ Board()
- AddCell(Cell c)
+ FindCell(int coordinate): Cell
+ FindCell<Type>(): Cell
+ ClickCell(Point2D pt): Cell
+ Load(string filename)
+ Draw() <<override>>
+ OnClick(EventArgs e)
+ CellNumber: int <<property, readonly>>
+ Width: int <<property, readonly>>
+ CellFactory: CellFactory <<property, readonly
+ WolrdCupCell: BuyableCell <<property>>
+ WorldCupSymbol: Bitmap <<property>>
+ IsAt(Point2D point): bool
+ ClickEvent : EventHandler

Figure 6 Example of Observer Pattern

The Observer pattern in this example enables the user object to alert other service objects to state changes. Other objects receive events of relevance from the MouseInputManager. The NotifyObservers() function in MIManager alerts all observers to process all click events simultaneously. All clickevent data is sent to the MouseClickedEvent interface and then to the MouseInputManager when a new event begins. The software will then be notified to call the NotifyObservers() function and wait for the action to take place. This programme requires the usage of an Observer Pattern since it will monitor the effectiveness of Click Events. For instance, when a user clicks on a Button Class, they may do activities like rolling dice, making a payment, or constructing a house. For Board Class, the user will interact with the Board's Cells using the mouse, such as clicking on a cell to display information clearly or changing which cell the World Cup will be organised in.