

## Design Overview for <<Gunny>>

Name: Dang Nam Khanh

Student ID: 104225661

### Summary of Program

This program is an interesting two-player game, two people will play two separate characters. Each character will have an inventory that contains weapons, special items such a teleportation, HP healing potions, etc.... Two players will use their weapons to shoot bullets at the opponent, if a player gets hit by a bullet, they will receive a deduction of HP. This is a turn-based game, a player will need to wait for the opponent's turn after their turn. They can use some special ability to enhance or modify the property of their bullets in turn. The bullets path will be influenced by multiple factors so that player will need an accurate bullet shooting. The game ends when one player has no HP left. Players will fight for 3 rounds, who gets two rounds first win the game.

### Required Roles

Describe each of the classes, interfaces, and any enumerations you will create. Use a different table to describe each role you will have, using the following table templates.

Table 1: <<role name>> details – duplicate

#### Player

Responsibility	Type Details	Notes
	Field type, parameter and return types	
<b>hitBox</b>	List<double> method	Define player's hitbox (not sure if I will use this)
<b>Top</b>	Double property	Define the dimension(top) of player
<b>Bottom</b>	Double property	Define the dimension(bottom) of player
<b>Left</b>	Double property	Define the dimension(left) of player
<b>Right</b>	Double property	Define the dimension(right) of player
<b>PlayerMove</b>	Void method	Allow user to move the player
<b>HP</b>	Int method	Return player's HP
<b>Mana</b>	Int method	Return player's mana

<b>Draw</b>	Void method	Draw player's image or shape that represents player hit box.
<b>DieOrAlive</b>	Boolean method	Check if player runs out of HP or not(die or alive)
<b>GetHit</b>	Void method	Decrease player's health if get hit by the bullet
<b>Shoot</b>	Boolean method	Fire the bullet
<b>New add method/property</b>		
<b>Rectangle</b>	Rectangle property	The rectangle represent the player's area for checkcollide
<b>CheckFall</b>	Bool method	Check if player stands on a ground
<b>CheckCollide</b>	Void method	Check if player collide with walls
<b>Bullet</b>	Bullet method	Return player's current shot out bullet
<b>Shooted</b>	Bool method	Check if player shoot on its turn
<b>CheckForFly</b>	Bool method	Special check collision for teleport bullet
<b>Update</b>	Void method	Check and update multiple condition of player
<b>CheckInvisible</b>	Void method	Modify Invisible state
<b>DelayTurn</b>	Void method	Used for shooting bullet
<b>DrawState</b>	Void method	Draw the freeze effect
<b>GameBlocks</b>	Void method	Add the list of all players
<b>ResetCoolDown</b>	Void method	Reset cooldown of items
<b>UpdateListOfRecord</b>	Void method	Update the text
<b>Angle</b>	Double property	Angle
<b>Bullet</b>	Bullet property	Return the bullet that player shot
<b>EnhancedBullet</b>	String property	Return string
<b>FinishTurn</b>	Bool property	Check if player finish its turn
<b>Force</b>	Double property	
<b>HP</b>	Double property	
<b>Invisible</b>	bool property	
<b>IsFreeze</b>	Bool property	
<b>IsTurn</b>	Bool property	
<b>NumberOfShots</b>	double property	
<b>NumberOfSuccessShots</b>	double property	
<b>OldHP</b>	Double property	
<b>PlayerArea</b>	Double property	

## Bullet

Responsibility	Type Details	Notes
	Field type, parameter and return types	
<b>CheckCollide</b>	Boolean method	Return true if the bullet hits the player, also trigger GetHit method from player
<b>ChangeColor</b>	Void method	Change color based on selected enhancement items
<b>Explode</b>	Void method	Define player's hitbox (not sure if I will use this)
<b>CameraIn</b>	Void method	Recenter the camera when the bullet is fired.
<b>Draw</b>	Void method	Draw the bullet shape and color
<b>New add method/property</b>		
<b>Rectangle</b>	Rectangle property	The rectangle represents the Bullet's area for checkcollide
<b>PlayerList</b>	List<Player> method	Create a list of players
<b>Stop</b>	Bool method	Return the stop method when the bullet collides
<b>Update</b>	Void method	Check multiple bullet states
<b>Algorithm</b>	Void method	Calculate the road of bullet
<b>DrawExplode</b>	Void method	Draw explosion effect
<b>RemoveGround</b>	Void method	Remove the ground
<b>RemovePlayer</b>	Void method	Remove player from the inner list
<b>Update</b>	Void method	Update multiple conditions of bullet

## Ground

Responsibility	Type Details	Notes
	Field type, parameter and return types	
<b>Draw</b>	Void method	Draw the bullet shape and color
<b>New add method/property</b>		
<b>Update</b>	Void method	Check multiple states of ground
<b>Bottom</b>	Double method	
<b>Destroyed</b>	Bool property	
<b>Left</b>	Double property	
<b>Rectangle</b>	Rectangle property	
<b>Right</b>	double property	
<b>Top</b>	Double property	

## GameBlock

Responsibility	Type Details	Notes
<b>X</b>	Double property	Return the x position of the object
<b>Y</b>	Double property	Return the y position of the object
<b>Draw</b>	Void method	Draw the object

## Item

Responsibility	Type Details	Notes
<b>OnClick</b>	Boolean	Return true if onclick.
<b>Description</b>	Void method	Return the Description
<b>FullDescrip</b>	String method	Return the Full Description
<b>Use</b>	Void method	User uses the item
<b>Draw</b>	Void method	Draw the Item

## Button

Responsibility	Type Details	Notes
<b>OnClick</b>	Boolean	Return true if onclick.
<b>Description</b>	Void method	Return the Description
<b>FullDescrip</b>	String method	Return the Full Description
<b>Open</b>	Void method	Open the the specific function that the button applies
<b>Draw</b>	Void method	Draw the button
<b>New add method/property</b>		
<b>IsMouseInRectangle</b>	Bool method	Check if mouse in rectangle
<b>OnClickButton</b>	Bool method	True if player click button
<b>GetState</b>	String method	Return the Gamestate

## ScoreManager

Responsibility	Type Details	Notes
<b>FirstPlayer</b>	Int property	Return first player score
<b>SecondPlayer</b>	Int property	Return second player score
<b>Round</b>	Int property	Return the passed round
<b>Draw</b>	Void method	Draw the score head bar

## GameManager

Responsibility	Type Details	Notes
<b>Turn</b>	String property	Check the turn of the players
<b>GameStart</b>	Void method	Start the game

<b>GameEnd</b>	Void method	Check if any player wins the matchup
<b>Playsound</b>	Void method	Play the needed sound in specific circumstances
<b>ResetGame</b>	Void method	Make a new matchup or play again matchup
<b>GetGameState</b>	Void method	Get the game state
<b>Update</b>	Void method	Check multiple state of program
<b>Draw</b>	Void method	Draw necessary UI

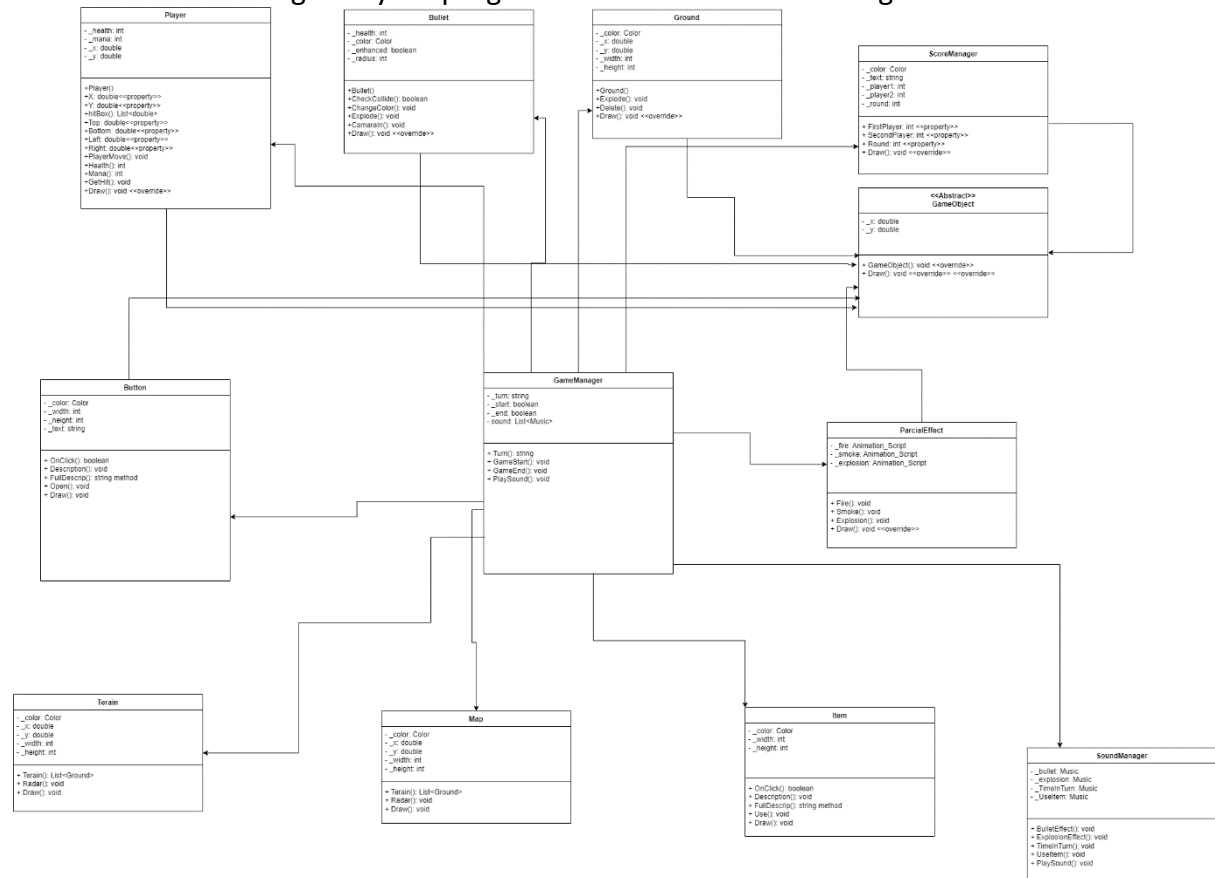
Table 2: <<enumeration name>> details

## Enum DrawOptions

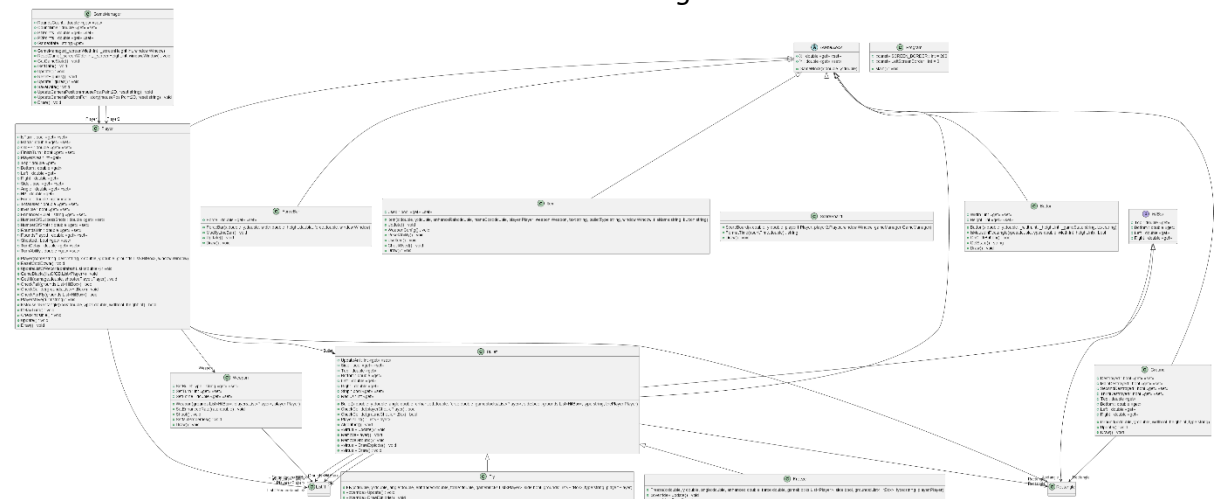
Value	Notes
<b>Dest</b>	Window, draw the menu
<b>AnchroOffsetX</b>	double, launch the game
<b>AnchroOffsetY</b>	double, paused the game
<b>ScaleX</b>	double, end the game, return to the menu
<b>ScaleY</b>	double
<b>Angle</b>	double
<b>FlipY</b>	bool

# Class Diagram

Provide an initial design for your program in the form of a class diagram.



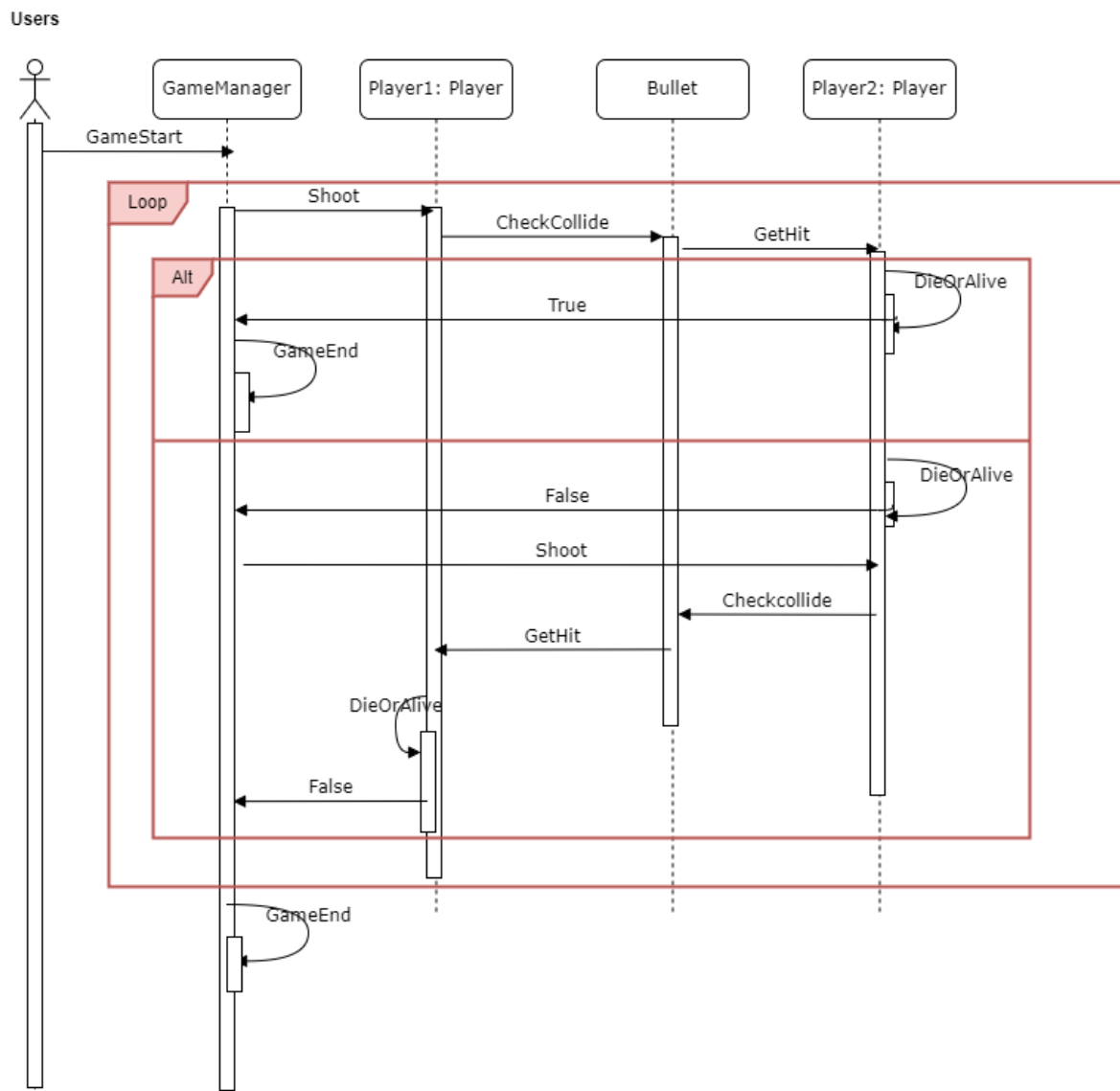
Initial UML Diagram



Final Diagram

The link to the UML for better visual: [6.2D - draw.io \(diagrams.net\)](https://draw.io/diagrams.net)

## Sequence Diagram



Provide a sequence diagram showing how your proposed classes will interact to achieve a specific piece of functionality in your program.

### The use of abstraction

The abstract class "GameBlock" has an abstract method "Draw()" that each class inherited from that class will be able to has its own Draw() method (each draw method will be different from each class) while it has the fields and properties such a "double X", "double Y".

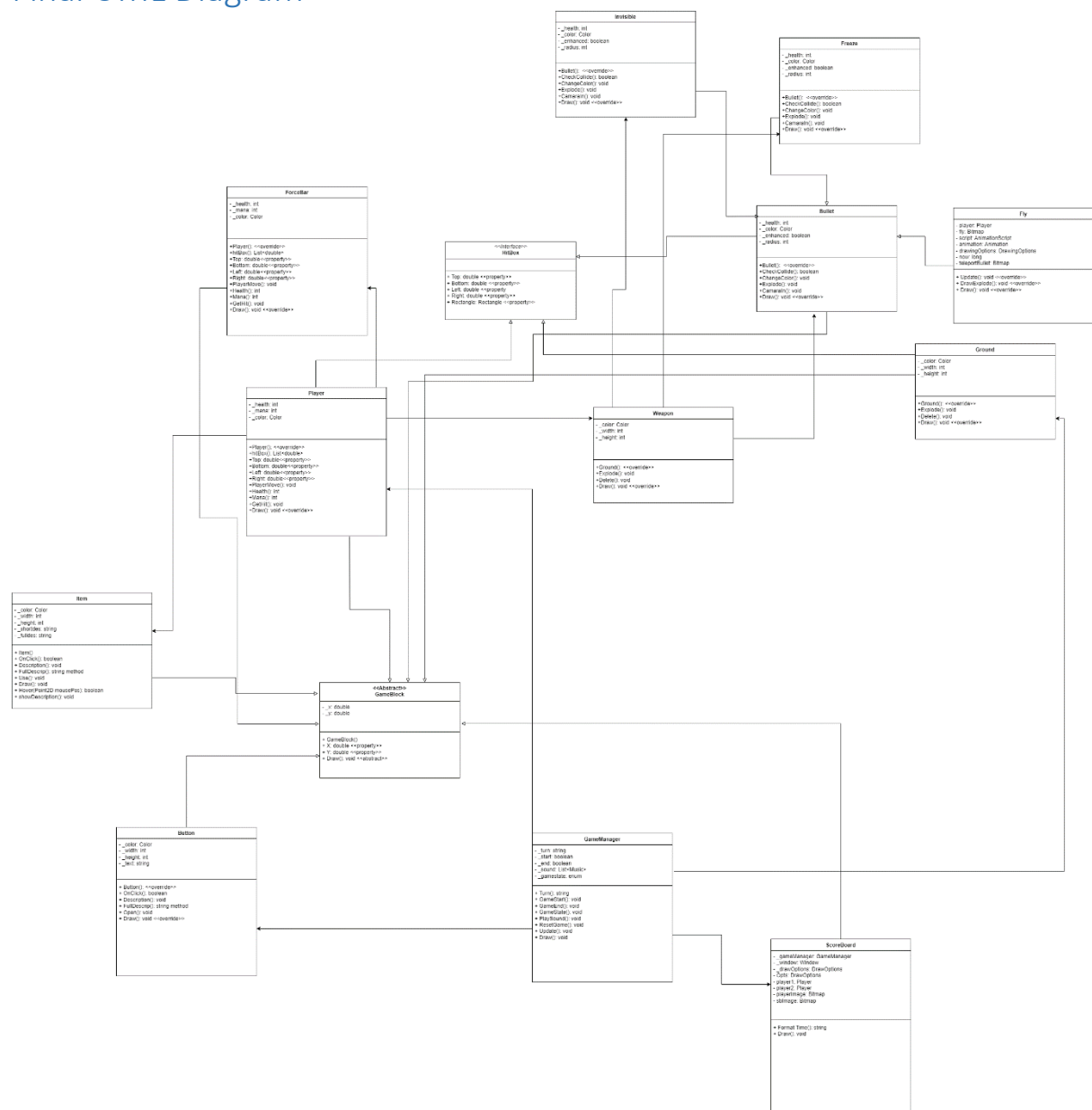
This abstraction emphasis on hiding details of the class and only show to the coder what "classes do" rather than "how these classes do it". It helps in building modular, maintainable, and loosely coupled code.

## The use of inheritance and polymorphism

There are several different classes that inherited from the GameBlocks class. Therefore, they will be able to be treated as GameBlock. Through polymorphism, I can write code that can work with objects of multiple types without needing to know their specific class.

In the previous example of abstraction, I implement a “CheckCollide” method in “Bullet” class. This method checks if there is any collision between object, it takes in various types of objects that have coordinates (X, Y) and its dimension. As all of object that has coordinates will be inherited from the GameBlock (class Player, class Bullet, class Ground), the “CheckCollide” will be able to process without considering about the types of class, it treats all the parameter as “GameBlock”.

## Final UML Diagram



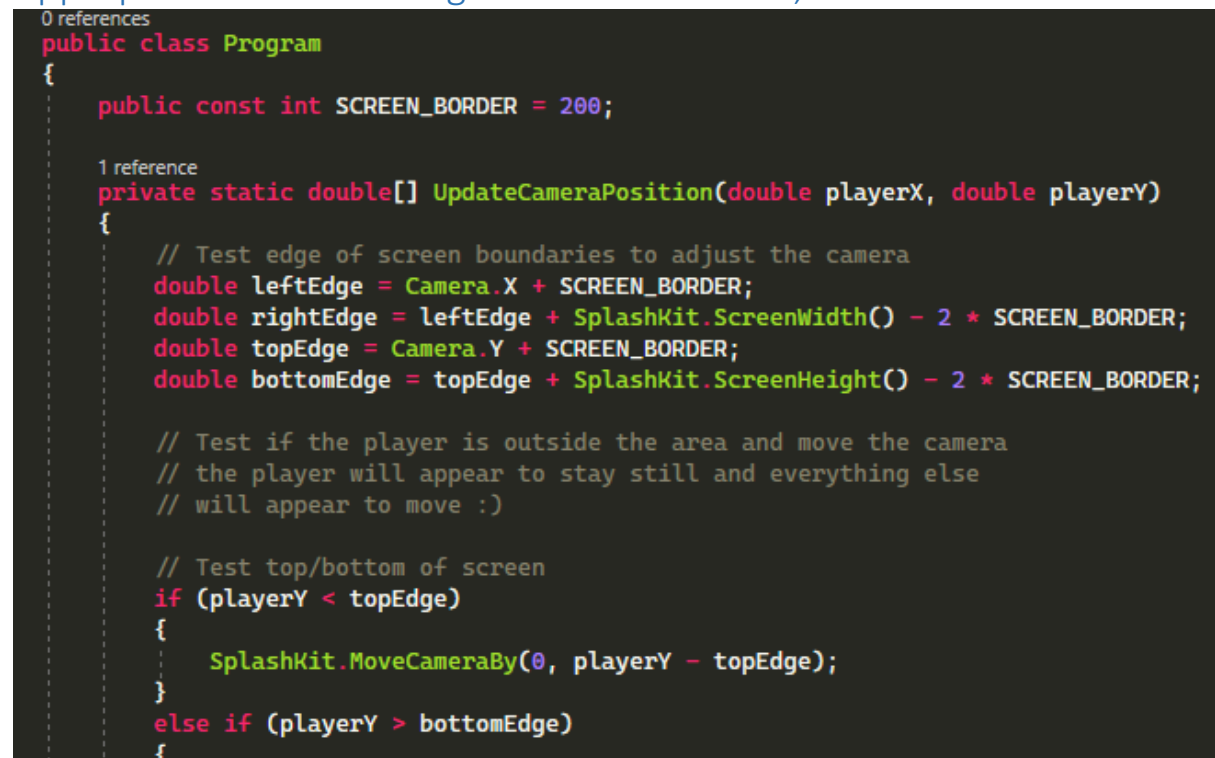
In this diagram, the “GameBlock” class is an abstract class, classes inherited from it are:



- Player
- Bullet
- Ground
- ScoreBoard
- Button
- Item

Overall, these classes will have the coordinates (x, y), also this helps a lot in terms of easier processing collision by "CheckCollide". The "GameManager" will be in charge of running and managing the game by using all defined classes.

## Appropriate use C# coding conventions - case, indentation



```
0 references
public class Program
{
    public const int SCREEN_BORDER = 200;

    1 reference
    private static double[] UpdateCameraPosition(double playerX, double playerY)
    {
        // Test edge of screen boundaries to adjust the camera
        double leftEdge = Camera.X + SCREEN_BORDER;
        double rightEdge = leftEdge + SplashKit.ScreenWidth() - 2 * SCREEN_BORDER;
        double topEdge = Camera.Y + SCREEN_BORDER;
        double bottomEdge = topEdge + SplashKit.ScreenHeight() - 2 * SCREEN_BORDER;

        // Test if the player is outside the area and move the camera
        // the player will appear to stay still and everything else
        // will appear to move :)

        // Test top/bottom of screen
        if (playerY < topEdge)
        {
            SplashKit.MoveCameraBy(0, playerY - topEdge);
        }
        else if (playerY > bottomEdge)
        {
        }
    }
}
```

The figure below shows the conventions in C# in my program:

- Case:
  - The use of PascalCase in Constant variable "SCREEN\_BORDER"
  - The use of PascalCase in Constant variable "SCREEN\_BORDER"
  - Class name, method name, variables use PascalCase as well.
- Indentation:
  - Each level of indentation uses four spaces.
  - The opening and closing braces for namespaces, classes, methods, and control structures (if) are aligned vertically.
  - The code within a method is indented inside the method block.
  - The code within a control structure (if) is indented inside the structure block.

## Code documentation

```
public class Program
{
    public const int SCREEN_BORDER = 200;

    1 reference
    private static double[] UpdateCameraPosition(double playerX, double playerY)
    {
        // Test edge of screen boundaries to adjust the camera
        double leftEdge = Camera.X + SCREEN_BORDER;
        double rightEdge = leftEdge + SplashKit.ScreenWidth() - 2 * SCREEN_BORDER;
        double topEdge = Camera.Y + SCREEN_BORDER;
        double bottomEdge = topEdge + SplashKit.ScreenHeight() - 2 * SCREEN_BORDER;

        // Test if the player is outside the area and move the camera
        // the player will appear to stay still and everything else
        // will appear to move :)

        // Test top/bottom of screen
        if (playerY < topEdge)
        {
            SplashKit.MoveCameraBy(0, playerY - topEdge);
        }
        else if (playerY > bottomEdge)
        {
            SplashKit.MoveCameraBy(0, playerY - bottomEdge);
        }

        // Test left/right of screen
        if (playerX < leftEdge)
        {
            SplashKit.MoveCameraBy(playerX - leftEdge, 0);
        }
        else if (playerX > rightEdge)
        {
            SplashKit.MoveCameraBy(playerX - rightEdge, 0);
        }
        double[] cameraPos = { leftEdge, rightEdge, topEdge, bottomEdge};
        return cameraPos;
    }
}
```

- In this example the first comment was for checking the edge of the camera.
- The second comment is for updating the camera when the player moves out of the camera range.

## Structured programming principles

```
1 reference
public Item Fetch(string id)
{
    foreach (Item item in _items)
    {
        if (item.AreYou(id))
        {
            return item;
        }
    }
    return null;
}
```

In this example, the code uses foreach loop and if loop:

- The foreach loop iterates through all items inside the player's inventory.
- The if condition check if the current item on the loop is the item that we want to use.

## Screen shot of program



The Menu options

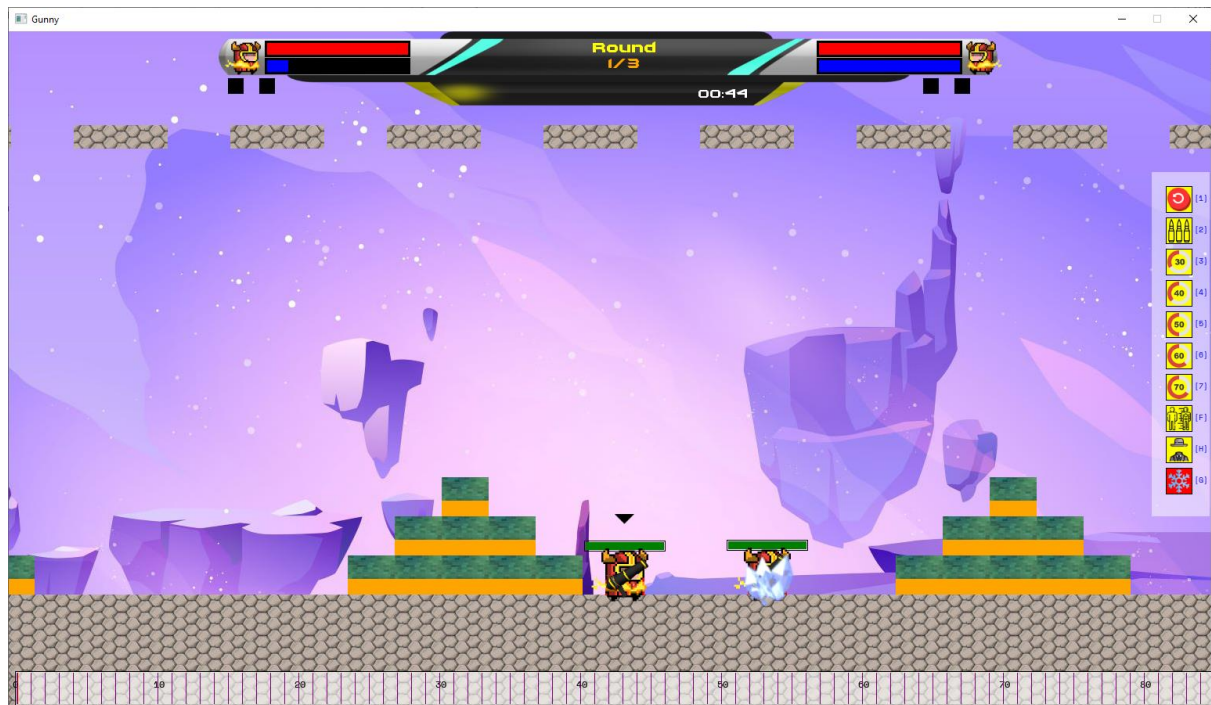


In game display



Player1 is Shooting

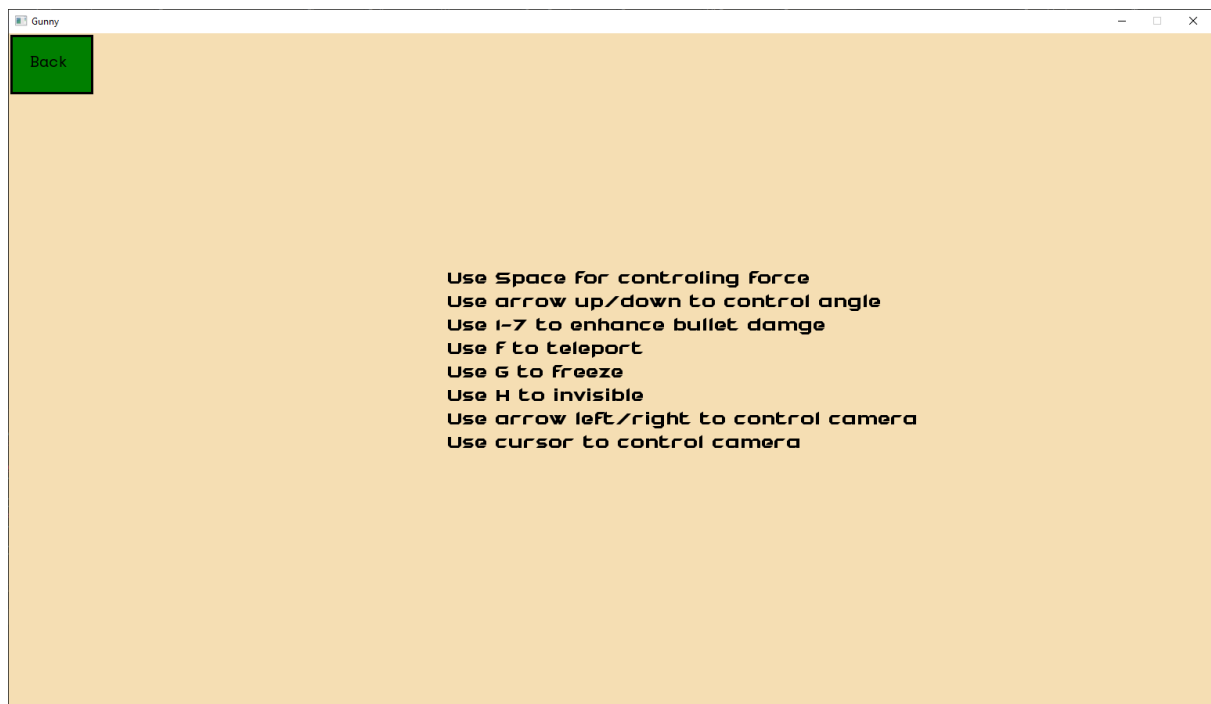




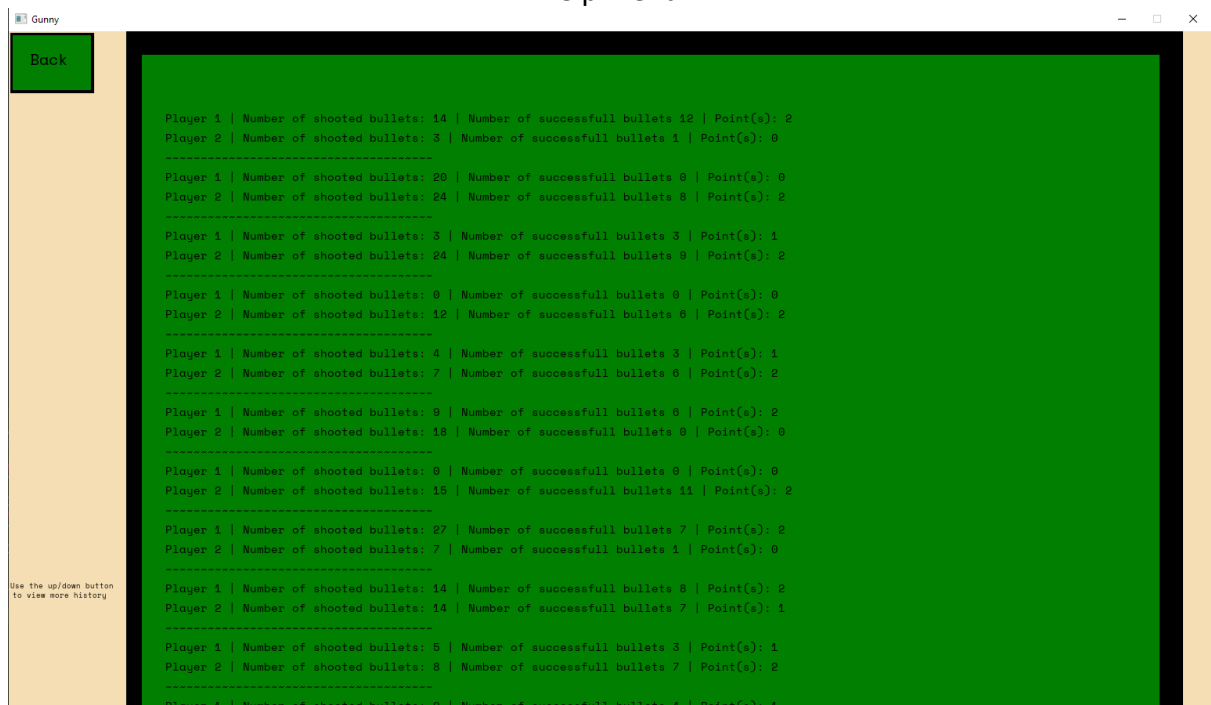
Three special abilities: freeze, invisible, teleport. They have cooldowns



Users can move camera to other places



## Help menu



## History menu

## Updated my design report.

New class created: Fly, Freeze, ScoreBoard

There are 3 main classes:

- GameManager is used to run the game.
- The Player class uses Shoot and GetHit to check if player's get hit by the bullet.
- The bullet's main method is "CheckCollide", it checks if the bullet hits the map or the player.

## Design pattern

### Singleton

The Singleton Design Pattern ensures that a class has only one instance and provides a global point of access to that instance. In my Gunny game project, I want to use a singleton for the GameManager class to ensure there is only one instance of the game manager throughout the game's execution. This class will manage the entire program and make sure that each section of the menu will be run after user clicked on. Moreover, it can even create new game for users in case that want to play again. GameManager can also reset the game state, and game stats. GameManager are also in charge of connecting the program Main method which holds the SplashKit GUI, it can receive some parameter from the Main Method and return some of values.

```
public GameManager(int _screenWidth, int _screenHeight)
{
    //set up the game as well as reset the game when play again
    ResetGame(_screenWidth, _screenHeight);
}

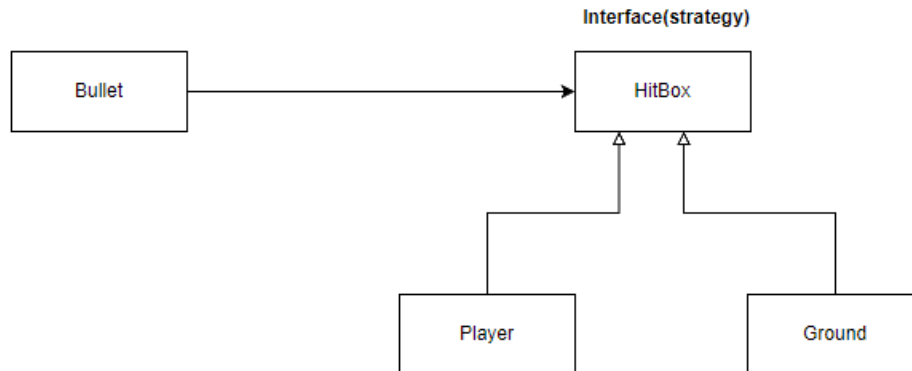
2 references
public void ResetGame(int _screenWidth, int _screenHeight)
{
    grounds = new List<HitBox>();
    for (int i = 0; i < 100; i++)
    {
        grounds.Add(new Ground(i * 30, 800, 30, 50));
        for(int j = 1; j < 200; j++)
        {
            grounds.Add(new Ground(i * 30, 800 + j*50, 30, 50));
        }
    }

    player1 = new Player("dnk", "the darkness inside your heart", 20, 20, grounds);
    player2 = new Player("dnk", "the darkness inside your heart", 1000, 200, grounds);
    gameStart = true;
    turn = "p1";
    onShooting = true;
    gameState = "menu";
    playerButton = new Button(_screenWidth / 2 - 100, _screenHeight / 2 - 110, 200, 100, "game", "Play");
    helpButton = new Button(_screenWidth / 2 - 100, _screenHeight / 2 + 110, 200, 100, "help", "Help?");
    quitButton = new Button(_screenWidth / 2 - 100, _screenHeight / 2 + 110 + 300, 200, 100, "quit", "Rage Quit?");
    screenWidth = _screenWidth;
    screenHeight = _screenHeight;
    _gameblocks = new List<Player>();
    _gameblocks.Add(player1);
    _gameblocks.Add(player2);
    player1.GameBlocks(_gameblocks);
    player2.GameBlocks(_gameblocks);
    now = DateTime.UtcNow.Ticks;
    resetNow = true;
}
```



## Strategy design pattern

The Strategy Design Pattern is a behavioral pattern that enable objects to change the algorithms or behaviors at runtime by encapsulating them as interchangeable strategies, making the code flexible, extensible, and promoting composition over inheritance.



The Strategy Pattern allows me to add new strategies, in the future I can add more game objects and the Bullet can still work with them. Inside the Bullet class, there are CheckCollide method which receive in the List<HitBox> parameter. Whenever it needs to check for the Ground, the class just simply have to change the parameter and it will works fine, respectively to Player.

```
1 reference
public bool CheckCollide(List<HitBox> grounds)
{
    foreach (HitBox game in grounds)
    {
        if ((X ≥ game.Left) && (X ≤ game.Right) && (Y ≥ game.Top) && (Y ≤ game.Bottom))
        {
            grounds.Remove(game);
            Console.WriteLine("deleted");
            return true;
        }
    }
    return false;
}
```

## Template Design Pattern

Template Method Pattern defines the skeleton of an algorithm in a method, allowing subclasses to implement specific steps. It promotes code reusability and extensibility while keeping the overall algorithm structure consistent across different implementations. In this HitBox interface, it implements the structure and each class that inherited from this interface would need to implement its own method/properties.

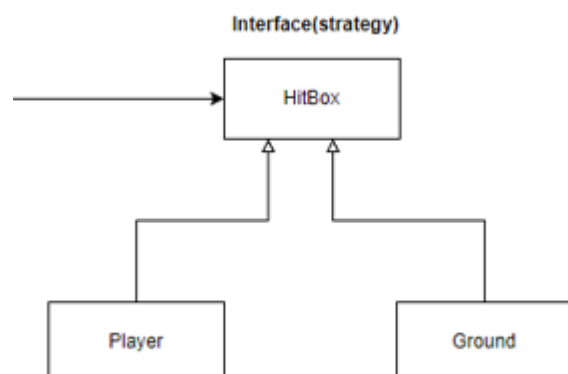
```
namespace Gunny
{
    18 references
    public interface HitBox
    {
        5 references
        public double Top
        {
            get;
        }

        3 references
        public double Bottom
        {
            get;
        }

        6 references
        public double Left
        {
            get;
        }

        6 references
        public double Right
        {
            get;
        }

        5 references
        public Rectangle Rectangle { get; }
    }
}
```



In my program, there is 1 more template method design, which is GameBlock, it has its has abstract method which is Draw() method, each subclass inherited from GameBlock will need to finish its own Draw() method.

```
9 references
public abstract class GameBlock
{
    private double _x;
    private double _y;
    //private double _width;
    //private double _height;

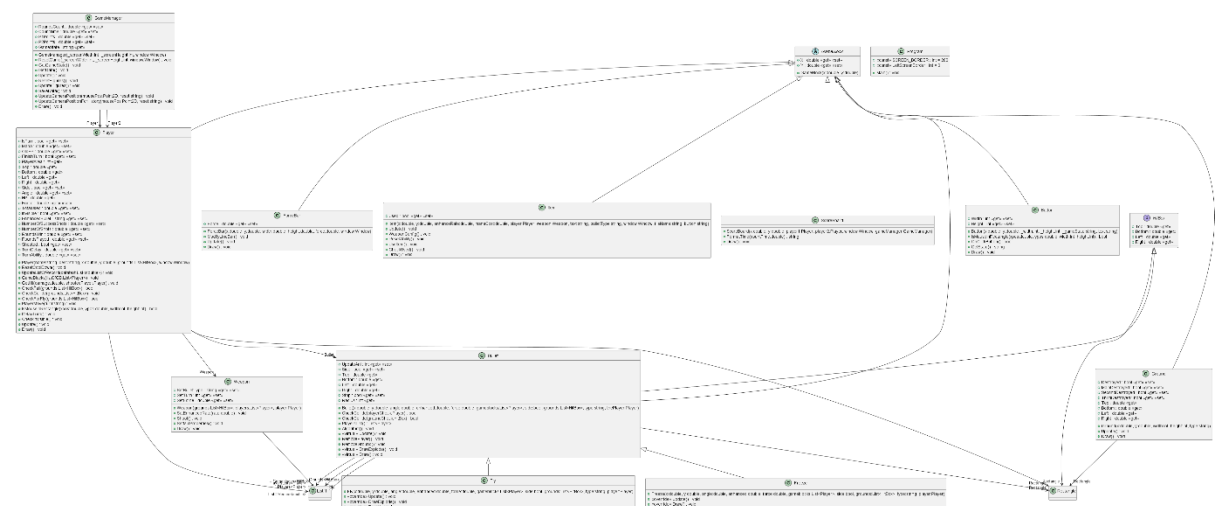
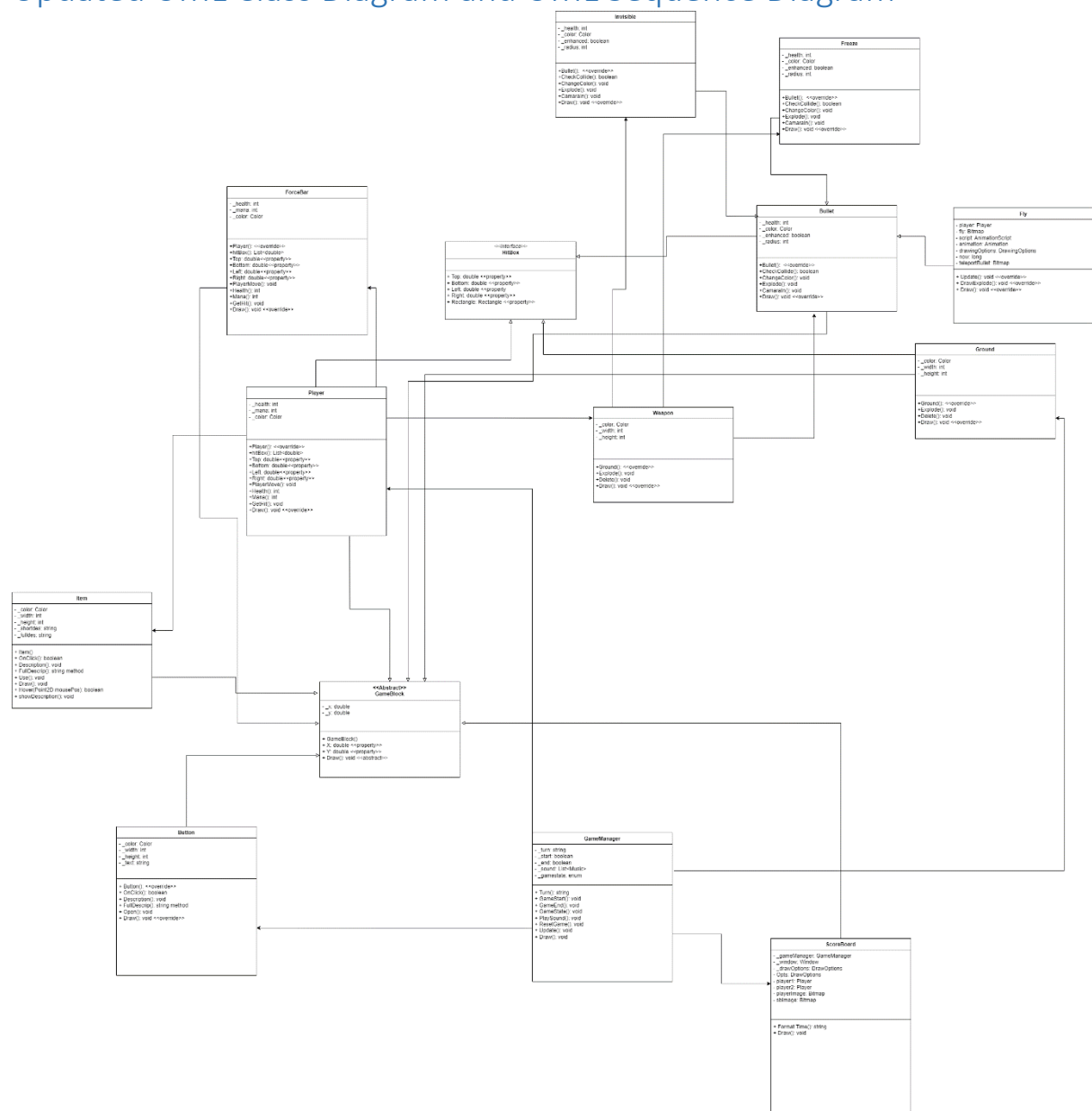
    4 references
    public GameBlock(double x, double y)
    {
        _x = x;
        _y = y;
    }

    37 references
    public double X
    {
        get { return _x; }
        set { _x = value; }
    }

    36 references
    public double Y
    {
        get { return _y; }
        set { _y = value; }
    }

    0 references
    public abstract void Draw();
}
```

## Updated UML Class Diagram and UML Sequence Diagram



*Zoom in for better visual.*



## Additional features and complexity

### Saving data

One of the features that makes my program become more complex than D-level is the saving data. Overall, as my game is not a high score game, it saves data based on player's identify so that it can be much easy to classify the data of players. I also did some data analysis based on the saved data. The saved data will be as following:

- Number of successful shots
- Number of shot bullets.
- Number of win rounds.

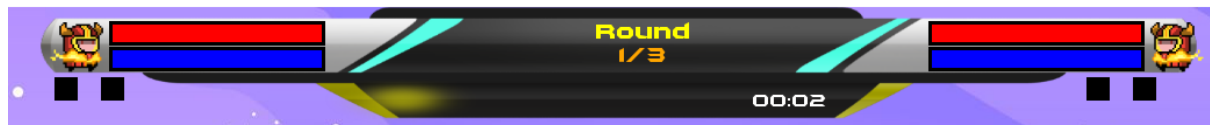
These data will serve data analysis, it is useful to balance the game and upgrade the game in the future.

```
Player 1 | Number of shooted bullets: 14 | Number of successfull bullets 12 | Point(s): 2  
Player 2 | Number of shooted bullets: 3 | Number of successfull bullets 1 | Point(s): 0  
-----
```

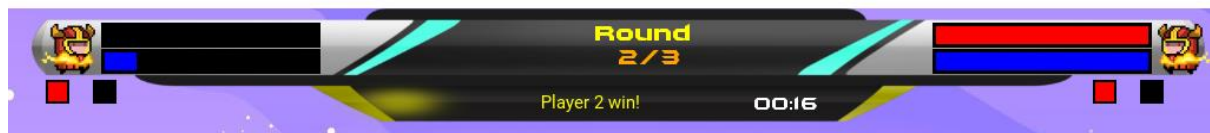
Recorded stats

### BO3

As my game is not about getting through levels, I implemented the three round fights. The players must win 2/3 rounds to win the game. Moreover, there will be a GameMatchup class to control the matchup instead of using GameManager. This will improve complexity and logical flows of the game. There is a scoreboard that will display the number of rounds, player HP/Mana, time and rounds win for each player.



Score board



Score board display win rounds and announce the winner of each round

### Multiplayer support

Two players will use the same buttons and UI/UX. But the great thing here is that there will be a camera point at player when it is that player's turn. This will enhance recognition from players so that they can have better control. I will be using OOP-UI classes to have better control and organize of the user interface.

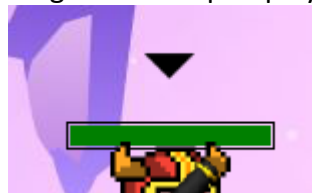
- ForceBar: users can use their mouse to mark the level they want to commit the force. This will enhance the accuracy which leads to accurate shots.
- Items: The frame containing the player's items will be displayed when it's their turn to shoot. The color of the items will display as the following rules:
  - Yellow: Available, can be used.
  - Gray: already used or not have enough Mana to use.
  - Red: on cooldowns, need to wait for several turns to use again.
- Mana system: Moving player will cause decrease of mana. Player uses mana for activating items. When the player runs out of mana, they can not move or use items. The mana is displayed at the top of the screen, inside the scoreboard.

- Supportive UI design: I draw the correct key for each item to the items frame.



Items frame and keys for each items

- Sign for turn: There will be a triangle at the top of player when it is that player's turn.



Sign

### Special abilities

Besides enhancing abilities such as shooting multiple turns, shooting three bullets in a turn and enhancing damage, there are some special abilities:

- Freeze: If this bullet hits the player, that player cannot shoot for the next two turns. Moreover, my program has the support for visual appearance, it shows the animation when the players are "freeze", along with the sound "ice cracking". It has cooldowns 3 turns.



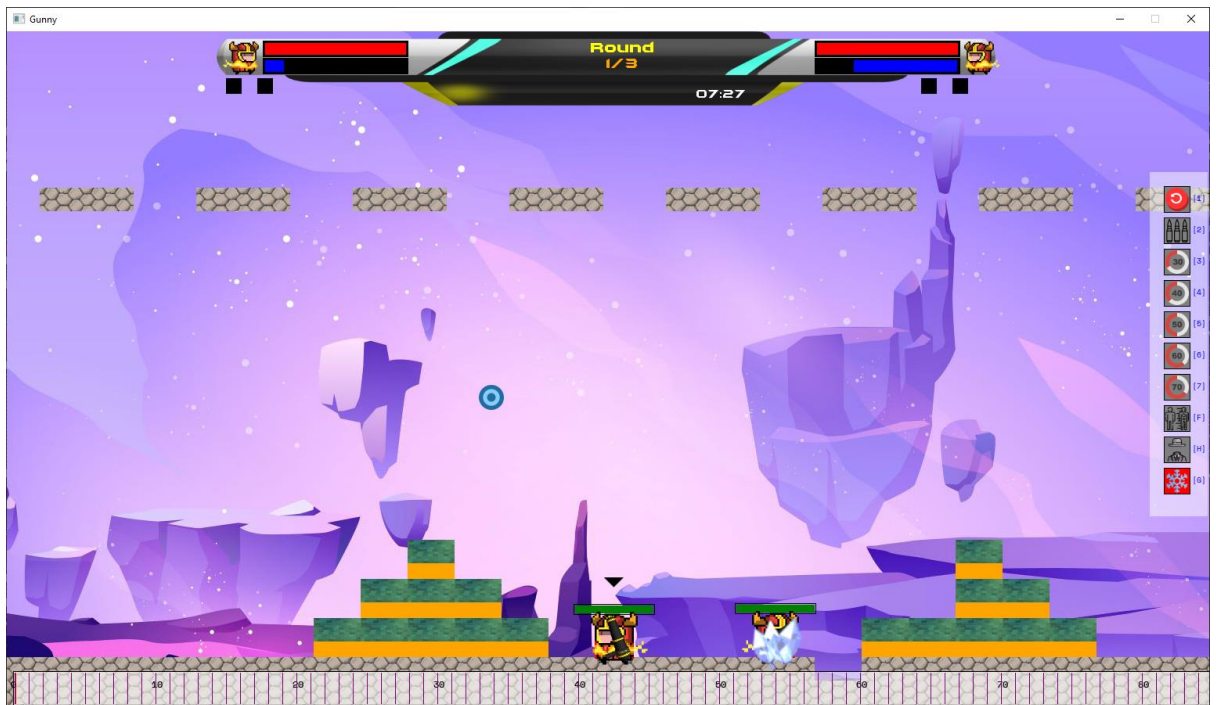
Exploding freezing effect



Being freeze effect

- Fly: The ability can be used to teleport to a certain place. It requires the player to shoot it, the player will be teleported to the place where the bullet lands. It has cooldown three turns. There are supported visual appearance for this ability which shows the blue fires where the player teleports to.





The teleport bullet.



Green fires where the bullet lands.

- Invisible: This ability will make the player's presence disappear. It will hide the player's teleportation bullet, but it does not hide the normal bullet(the damage causing bullets).

### Sound improvement

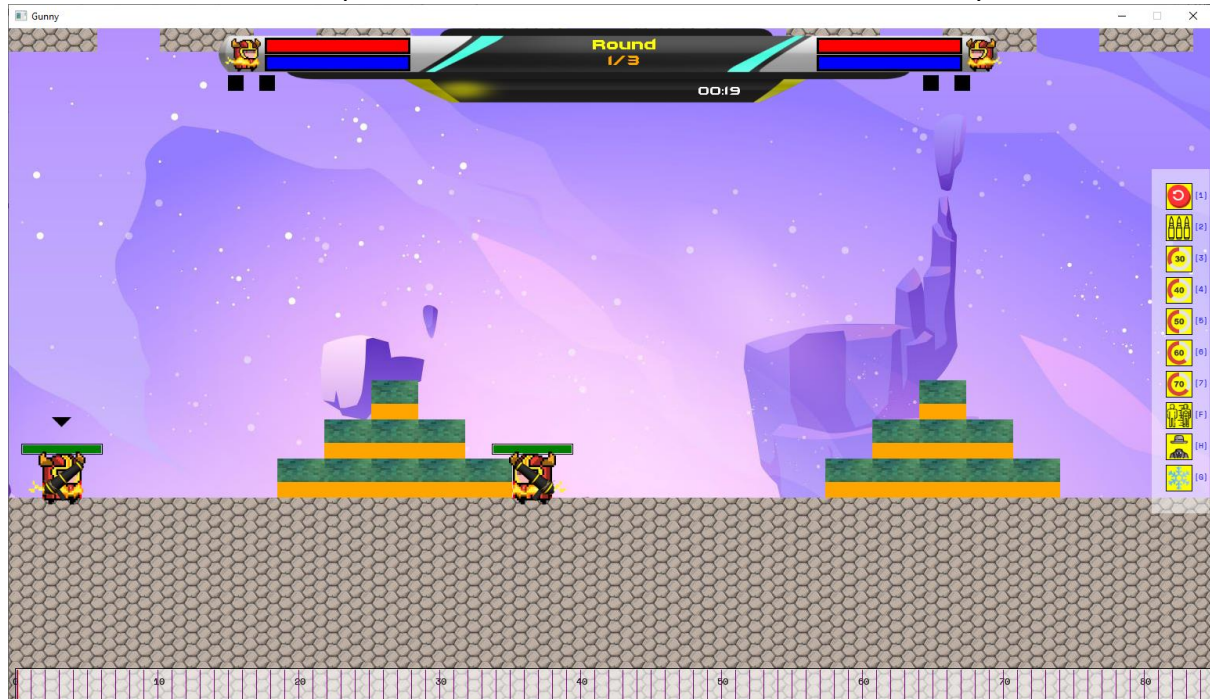
I have made some additional improvement for the sounds in the game:

- Sound of shooting: When the player shoots, there will be a small exploding sound
- Sound of bullet collision: When the bullets collide, there will be an exploding sound.
- Sound of Freeze abilities: The "Ice Cracking" sound when the bullet hits the player.

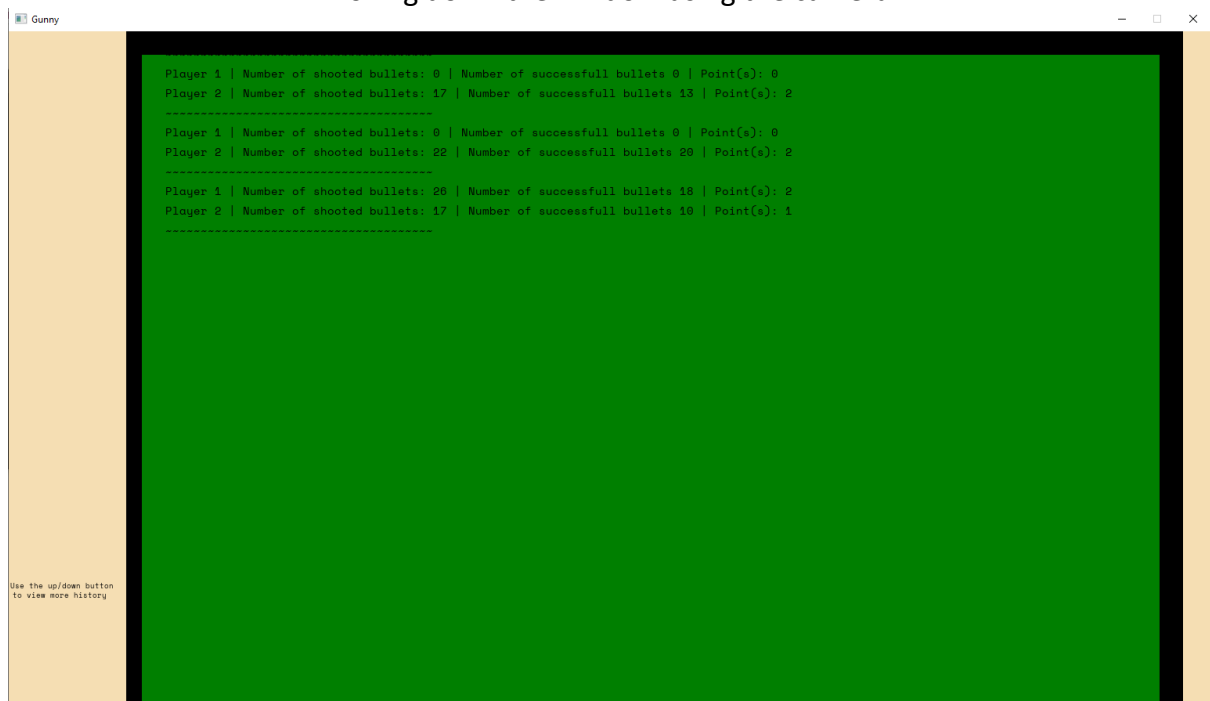
- Background music: I have modified the volume of this music so that it does not make too much noise.

### Camera Implementation

Overall, users can use either arrow key or mouse to control the camera, the camera will be at the center of the player when it is that player's turn. Moreover, in the history menu, users can use the arrow up/down to scroll down the list of recorded history.



Moving down the window using the camera.



Scrolling the history list.

## Loosing HP animation/effect

When the player loses HP, there will be a red HP that slowly reduces the current HP

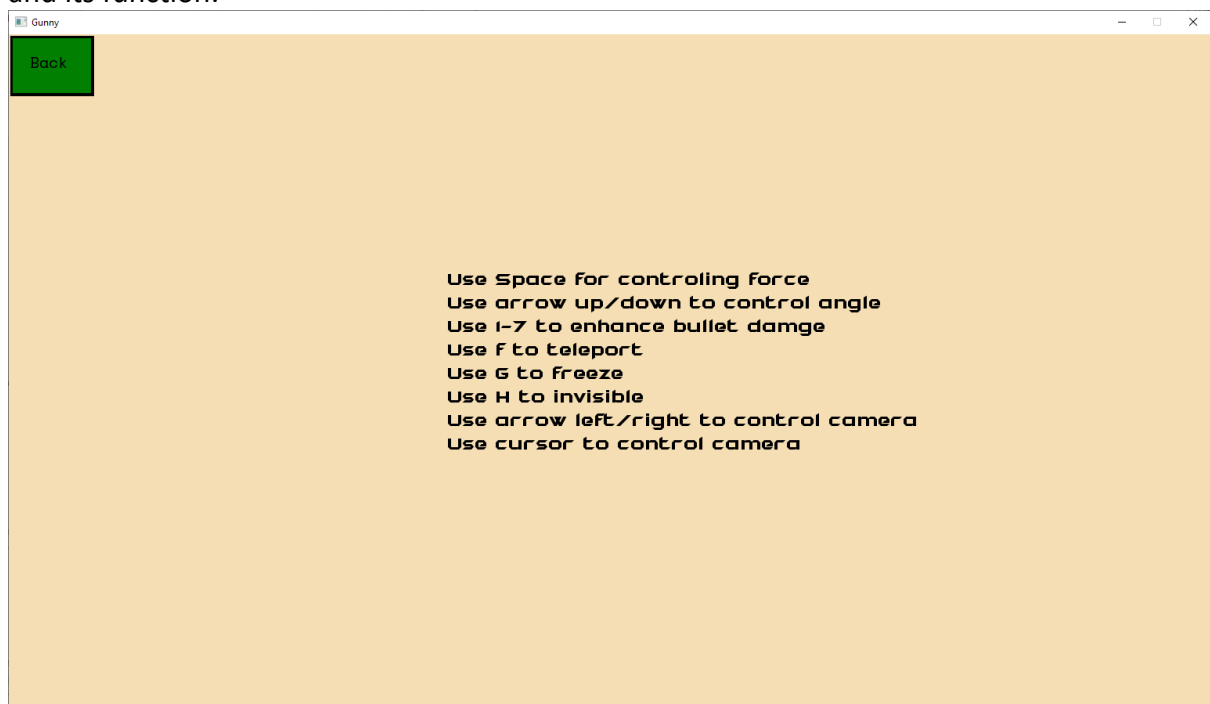


Loosing HP effect

Looking at these upgrades, my program has a better complexity than the D-level program.

## Help menu

I created this menu to help users become familiar with the game. I added all the buttons and its function.



Keys for controlling the game

## Animation

- Button animation:



When not hover



When hover

- Bullet trace: Bullet trace consist of 5 elements, 4 circle and an image.



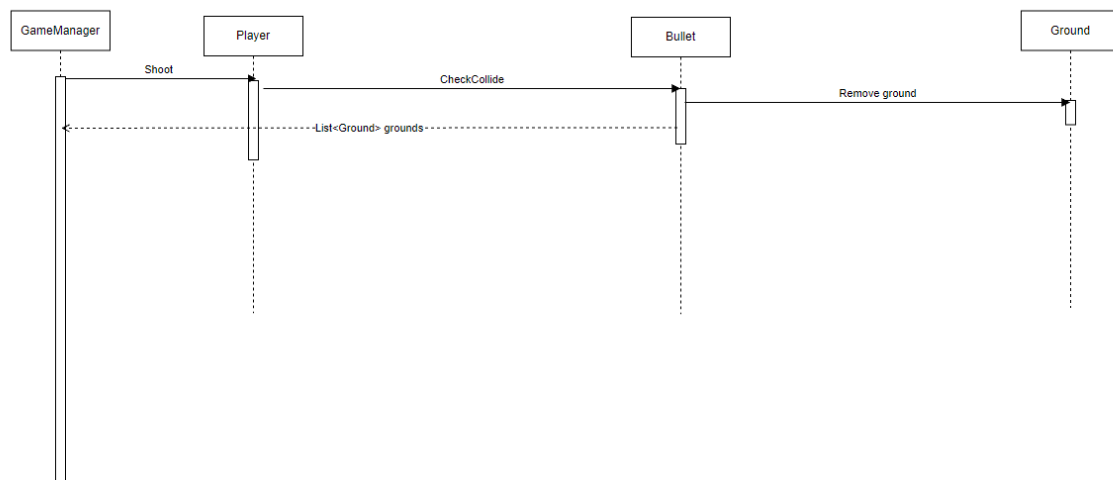
Bullet trace

- Blocks and ground: I added the Bitmap for each grounds and blocks, these grounds and blocks can be interact through the bullet



Blocks and grounds can be destroyed

## Additional UML Sequence Diagram



*Process of bullet colliding with ground.*

## How my program meets a very high standard

### Reasonable complexity

- Overall users have to carefully control multiple elements such as 'force', 'angle', 'enhanced abilities'.... All these elements require users to think before shooting bullet rather than just fire bullets without brains.
- My program also just not manipulate data, it also analysis it to generate realistic figures that helps player to monitor their play.
- The program also implemented with good OOP logic. In terms of UX/UI I have multiple objects what serves multiple function:
  - ForceBar: is used to display the force that player select to fire bullets.
  - Item: is used to either offer users abilities and show used/not used abilities.
  - Fly, Invisible, Freeze: same as Item but these abilities are completely different.
  - HP,Mana: show hp and mana
  - ScoreManager: Monitor score and points
  - Players, Bullet, Ground,.. all have its own task and interact with each other.

### Use of object-oriented design concepts

- In this project there are two main abstract class which are GameBlocks and Button, and HitBox interface. It promotes deduction of code duplication as well as provides better working with codes.
- Methods are shot, targeted and have meaningful actions and meaningful name.

```
1 reference
public bool CheckCollide(List<HitBox> grounds)
{
    foreach (HitBox game in grounds)
    {
        if ((X ≥ game.Left) && (X ≤ game.Right) && (Y ≥ game.Top) && (Y ≤ game.Bottom))
        {
            grounds.Remove(game);
            Console.WriteLine("deleted");
            return true;
        }
    }
    return false;
}
```

- I have applied all OOP concepts to my program:
  - Inheritance
  - Design patterns
  - Polymorphism
  - Encapsulation
  - Abstraction
  - Interface