Design Overview for <<Adventure Time>>

Name: Huynh Trung Chien
Student ID: 104848770

Summary of Program

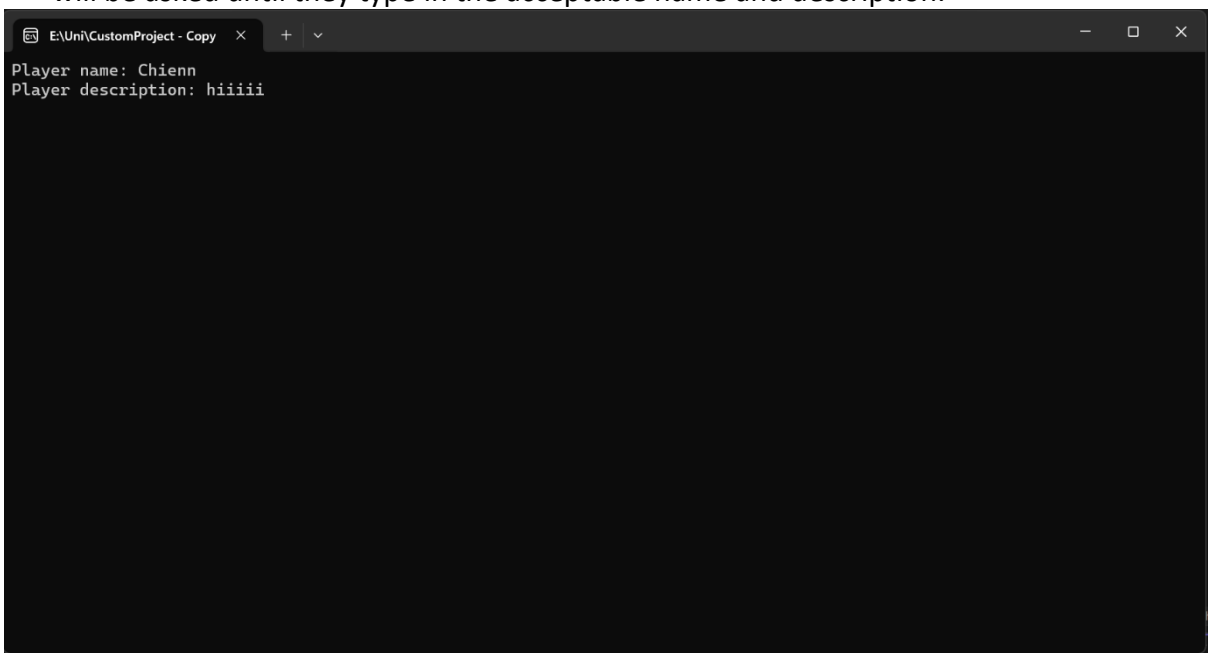
For my D custom program, I aimed to create an adventure game using the SplashKit library in C#.

First the users will be asked for their personal information before playing the game, after that the user will be directed to the game interface. In this game, the player will use the Up, Down, Right, and Left keys on their keyboard to move the character around the map. When moving around the player can collected the objects on the map such as golds, swords. There are also bombs in the maps, and if the players touch these bombs, they will lose the game. Then the players can restart the games by pressing the R key or quit the game by pressing the Q key. Moreover, the players also have their inventory, and they can check the inventory by using the Tab key.

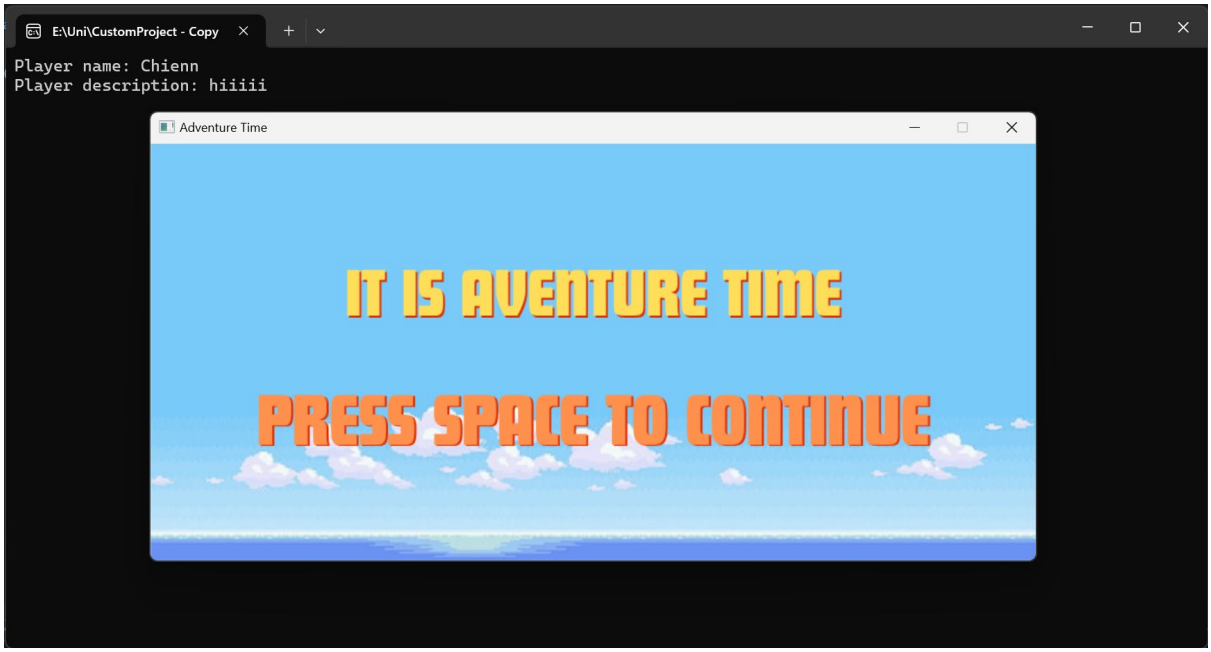
This is just the initial plan for my custom program, there will be changes when I develop this game in the future

Here are some first pictures of the game:

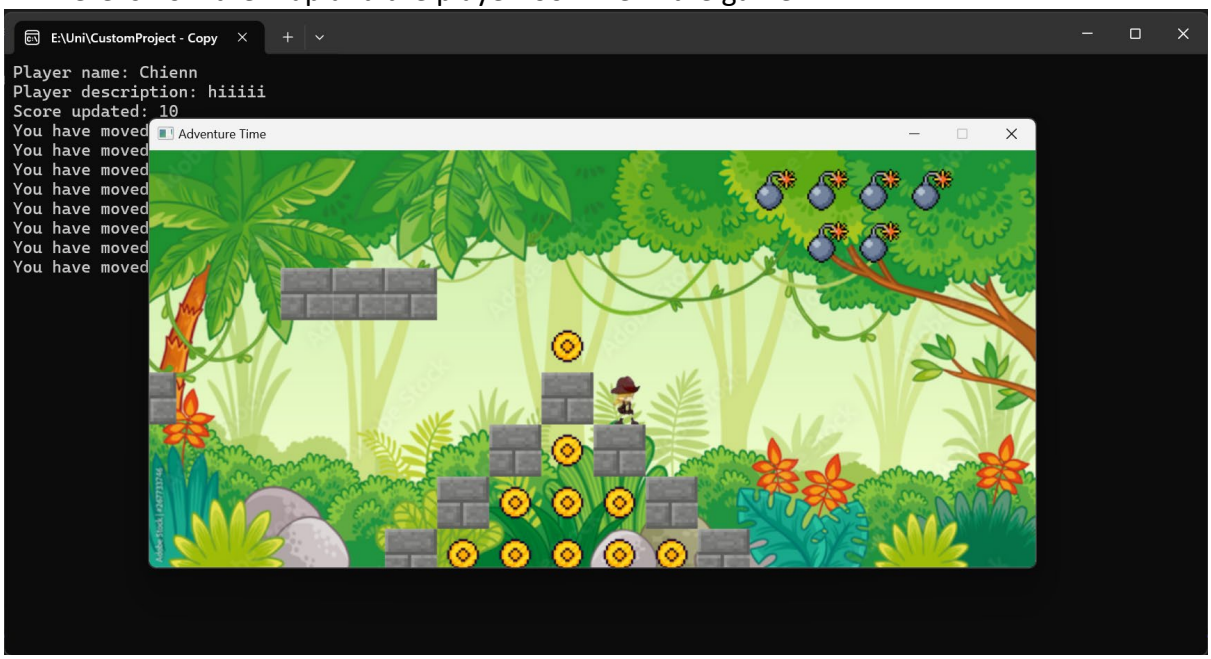
- This is the game requiring the player to type in their name before starting, the player will be asked until they type in the acceptable name and description.



- Here is the game interface:



- Here is how the map and the player look like in 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: <<IdentifiableObject>> details – Public class

Responsibility	Type Details	Notes
Fields	- _identifiers : List<string>	Identifiers to locate objects in game

Constructor	+ IdentifiableObject(string[] idents)	
Methods	+ AreYou(string id) : bool	Check if the object is itself
	+ FirstId() : string	Return the first id of the objects

Table 2: <<GameObject : IdentifiableObject>> details – abstract class

Responsibility	Type Details	Notes
Fields	- _description : string	The object's description
	- _name : string	The object's name
Constructor	+ GameObject(string[] ids, string name, string description)	
Properties	+ Name : string <<readonly property>>	
	+ ShortDescription : string <<readonly property>>	Provide short description
	+ FullDescription : string <<readonly property>>	Provide the full description

Table 3: <<DrawableObject : GameObject>> details – abstract class

Responsibility	Type Details	Notes
Fields	# _xLocation : double # _yLocation : double # _image : Bimap	
Constructor	+ DrawableObject(double xLocation, double yLocation, string[] ids, string name, string description, string imagePath)	
Methods	+ Draw() : void <<virtual>>	Draw the objects on the map
Properties	+ X: double <<properties>> + Y: double <<properties>> + Image : Bitmap <<readonly properties>>	

Table 4: <<CollectibleBomb : DrawableObject>> details - public class

Responsibility	Type Details
Constructor	+ CollectibleBomb(double xLocation, double yLocation) : base(xLocation, yLocation, new string[] { "bomb" }, "Bombs!!", "Do not touch the bombs!", "Bomb.png")

Table 5: <<Gold : DrawableObject>> details – public class

Responsibility	Type Details
Constructor	+ CollectibleGold(double xLocation, double yLocation) : base(xLocation, yLocation, new string[] { "gold" }, "Golds!!", "Collect golds to win the game!!", "Gold.png")

Table 6: <<Land : DrawableObject>> details – public class

Responsibility	Type Details
Constructor	+ Land(double xLocation, double yLocation) : base(xLocation, yLocation, new string[] { "" }, "", "", "land.png")

Table 7: <<Sword : DrawableObject>> details – public class

Responsibility	Type Details
Constructor	+ Sword(double xLocation, double yLocation) : base(xLocation, yLocation, new string[] { "sword" }, "Metal Sword!", "Meelee weapons!", "sword.png")

Table 8: <<GameMap : GameObject>> details – public class

Responsibility	Type Details	Notes
Fields	- _mapWidth : int - _mapHeight : int - _background : Bitmap - _jungle : Bitmap - _sky : Bitmap - _sea : Bitmap - _lands : List<Land> - _bombs : List<CollectibleBomb> - _golds : List<CollectibleGold> - _sword : List<Sword> - _txtMap : string[]	
Constructor	+ GameMap(string[] ids, string name, string description)	
Methods	- ReadMapFromFile(string filename) : void	Reads the map from a text file and initializes _txtMap.
	- LoadBackgrounds() : void	Loads the background images.
	+ SetUpGameMap() : void	Sets up the game map based on the text map representation.
	+ Draw() : void	Draws the game map and its objects.
Properties	+ Width : int <<readonly property>>	Returns the width of the game map in pixels.
	+ Height : int <<readonly property>>	Returns the height of the game map in pixels.
	+ Lands : List<Land> <<readonly property>>	Returns the list of land objects.
	+ Bombs : List<CollectibleBomb> <<readonly property>>	Returns the list of collectible bombs.
	+ Golds : List<CollectibleGold> <<readonly property>>	Returns the list of collectible golds.
	+ Swords : List<Sword> <<readonly property>>	Returns the list of swords.
	+ Background : Bitmap <<property>>	Gets or sets the current background.

	+ Sea : Bitmap <<readonly property>>	Returns the sea background bitmap.
	+ Jungle : Bitmap <<readonly property>>	Returns the jungle background bitmap.
	+ Sky : Bitmap <<readonly property>>	Returns the sky background bitmap.

Table 9: <<Inventory>> details – public class

Responsibility	Type Details	Notes
Fields	- _items : List<Item>	Stores the list of items in the inventory.
Constructor	+ Inventory() : void	Initializes the inventory with an empty list of items.
Methods	+ HasItem(string id) : bool	Checks if an item with the given identifier is in the inventory.
	+ Put(Item itm) : void	Adds an item to the inventory.
	+ Fetch(string id) : Item	Retrieves an item with the given identifier from the inventory.
Properties	+ ItemList : string <<readonly property>>	Returns a string listing the short descriptions of all items in the inventory.

Table 10: <<IHaveInventory>> details - Interface

Responsibility	Type Details	Notes
Methods	Locate(string id) : GameObject	Locates and returns a GameObject with the given identifier from the inventory.
Properties	+ Name : string <<property>>	Gets the name of the inventory holder.

Table 11: <<Item : GameObject>> details – public class

Responsibility	Type Details	Notes
Constructor	+ Item(string[] idents, string name, string description)	Initializes the Item with identifiers, name, and description.

Table 12: <<CollisionHandler>> details – public class

Responsibility	Type Details	Notes
Fields	- _gameMap : GameMap	Stores the reference to the game map.
	- _player : Player	Stores the reference to the player.

Constructor	+ CollisionHandler(GameMap gamemap, Player player)	Initializes the collision handler with references to the game map and player.
Methods	+ CheckBombCollision() : void	Checks for collisions between the player and bombs, handles the collision action, and removes the bombs.
	+ IsPlayerOnGround() : bool	Checks if the player is on the ground by verifying collision with land.
	+ IsCollideWithLand(double x, double y) : bool	Checks for collisions between the player and land at the given coordinates.

Table 13: <<CameraHandler>> details – public class

Responsibility	Type Details	Notes
Fields	- _gameMap : GameMap	Stores the reference to the game map.
	- _player : Player	Stores the reference to the player.
Constructor	+ CameraHandler(GameMap gamemap, Player player)	Initializes the camera handler with references to the game map and player.
Methods	+ HandleCamera() : void	Adjusts the camera position based on the player's location and the boundaries of the game map.

Table 14: <<AnimationHandler>> details – public class

Responsibility	Type Details	Notes
Fields	- _player : Player	Stores the reference to the player.
Constructor	+ AnimationHandler(Player player)	Initializes the animation handler with a reference to the player.
Methods	+ HandleCamera() : void	Adjusts the camera position based on the player's location and the boundaries of the game map.
	- UpdateStandingAnimation() : void	Updates the player's standing animation.
	- UpdateRunningAnimation() : void	Updates the player's running animation based on key presses.
	- StartAnimationIfNotRunning(string animationName) : void	Starts the specified animation if it is not already running.
	+ UpdateDeadAnimation() : void	Updates the player's dead animation.

	+ Die() : void	Handles the player's death animation.

Table 15: <<Command : IdentifiableObject>> details – abstract class

Responsibility	Type Details	Notes
Fields	Inherits fields from IdentifiableObject	Stores the reference to the player.
Constructor	+ Command(string[] ids)	Initializes the command with identifiers.
Methods	+ Execute(Player p, string[] text) : string <<abstract>>	Abstract method to execute the command.

Table 16: <<LookCommand : Command>> details : public class

Responsibility	Type Details	Notes
Fields	Inherits fields from Command	No additional fields.
Constructor	+ LookCommand()	Initializes the LookCommand with the identifier "look".
Methods	+ Execute(Player p, string[] text) : string <<override>>	Executes the look command, returning the description of the specified object.
	- LookAtIn(string thingId, IHaveInventory container) : string	Looks for an item in the given inventory and returns its description.

Table 17: <<GameProcessor>> details : public class

Responsibility	Type Details	Notes
Fields	- _window : Window	Stores the game window.
	- _map : GameMap	Stores the game map.
	- _player : Player	Stores the player.
	- _playerName : string	Stores the player's name.
	- _interface : Bitmap	Stores the interface bitmap.
	- _playerName : string	Stores the player's name.
	- _playerDescription : string	Stores the player's description.
	- _gameStarted : bool	Indicates if the game has started.
	- _gameOver : bool	Indicates if the game is over.
	- _showingInstructions : bool	Indicates if the instructions are being shown.
Constructor	+ GameProcessor()	Initializes the game processor, setting up the game state.
Methods	+ Run() : void	Runs the main game loop.
	- GetUserName() : void	Looks for an item in the given inventory and returns its description.
	- GetUserDescription() : void	Prompts the user for their description.
	- InitializeMap() : void	Initializes the game map.

	- InitializePlayer() : void	Initializes the player.
	- HandleGameStart() : void	Handles the game start sequence, including showing instructions.
	- HandleGameOver() : void	Handles the game over sequence.
	- HandleGamePlay() : void	Handles the main gameplay sequence.
	- CheckDeathState() : void	Checks if the player is dead and handles game over state.
	- UpdateInventory() : void	Updates the player's inventory based on collected items.
	- OpenInventory() : void	Handles opening and displaying the player's inventory.
	- UpdateGameMap() : void	Updates the game map based on player actions.

Table 18: <<Player : DrawableObject, IHaveInventory>> details – public class

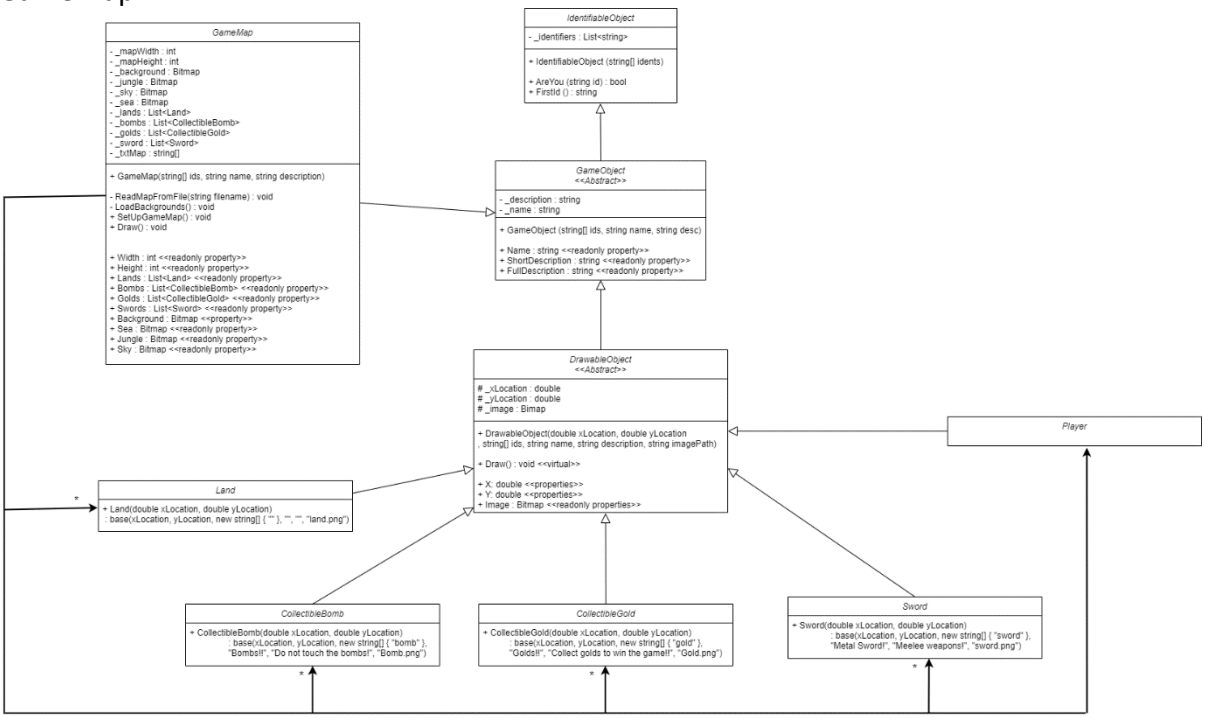
Responsibility	**Type Details**	**Notes**
Fields	- _window : Window	Stores the gravity effect on the player.
	- _playerSprite : Sprite	Stores the player's sprite.
	- _gameMap : GameMap	Stores the game map reference.
	- _isDead : bool	Indicates if the player is moving.
	- _isMoving : bool	Stores the interface bitmap.
	- _inventory : Inventory	Stores the player's inventory.
	- _collectedGolds : int	Stores the number of collected golds.
	- _collectedSwords : int	Stores the number of collected swords.
Constructor	+ Player(GameMap gameMap, string name, string description)	Initializes the player with the game map, name, and description.
Methods	+ Update() : void	Updates the player's state.
	+ Draw() : void <<override>>	Draws the player on the screen.
	- UpdateYLocation() : void	Updates the player's Y location considering gravity and collisions.
	- InitializePlayerSprite() : void	Initializes the player's sprite.
	- InitializePlayerState(GameMap gameMap) : void	Initializes the player's state.
	+ Locate(string id) : GameObject <<interface implementation>>	Locates an item in the player's inventory or the player itself.
Properties	+ Inventory : Inventory <<property>>	Gets or sets the player's inventory.

	+ Gravity : double <<property>>	Gets or sets the player's gravity.
	+ IsMoving : bool <<property>>	Gets or sets the player's dead state.
	+ IsDead : bool <<property>>	Updates the player's inventory based on collected items.
	+ GameMap : GameMap <<readonly property>>	Gets the game map reference.
	+ CollectedGolds : int <<property>>	Gets or sets the number of collected golds.
	+ CollectedSwords : int <<property>>	Gets or sets the number of collected swords.
	+ PlayerSprite : Sprite <<readonly property>>	Gets the player's sprite.

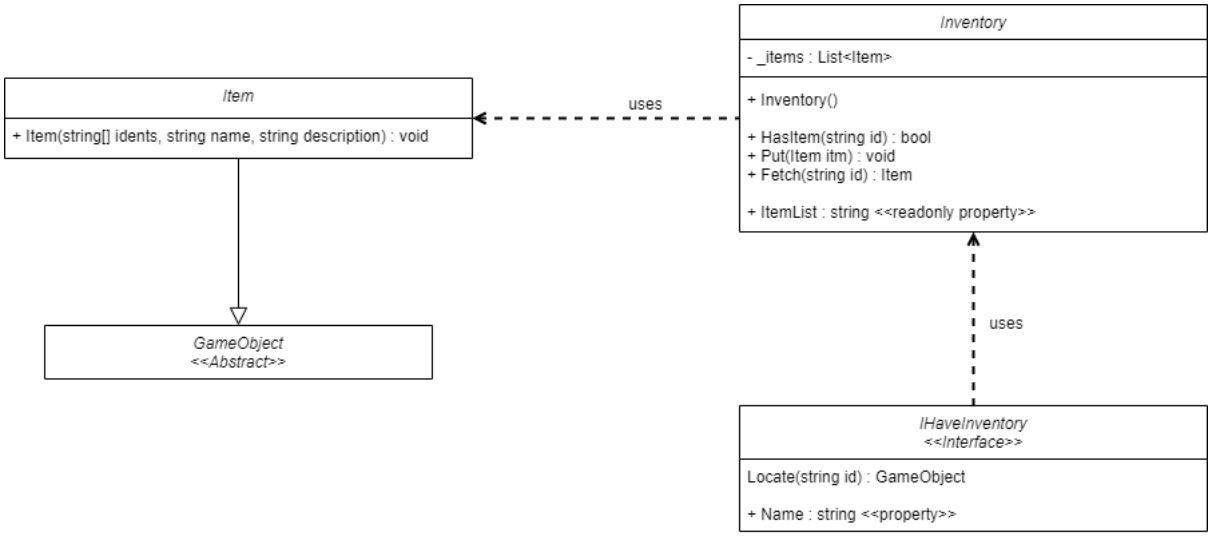
Class Diagram

Here is my initial design for my D custom program.

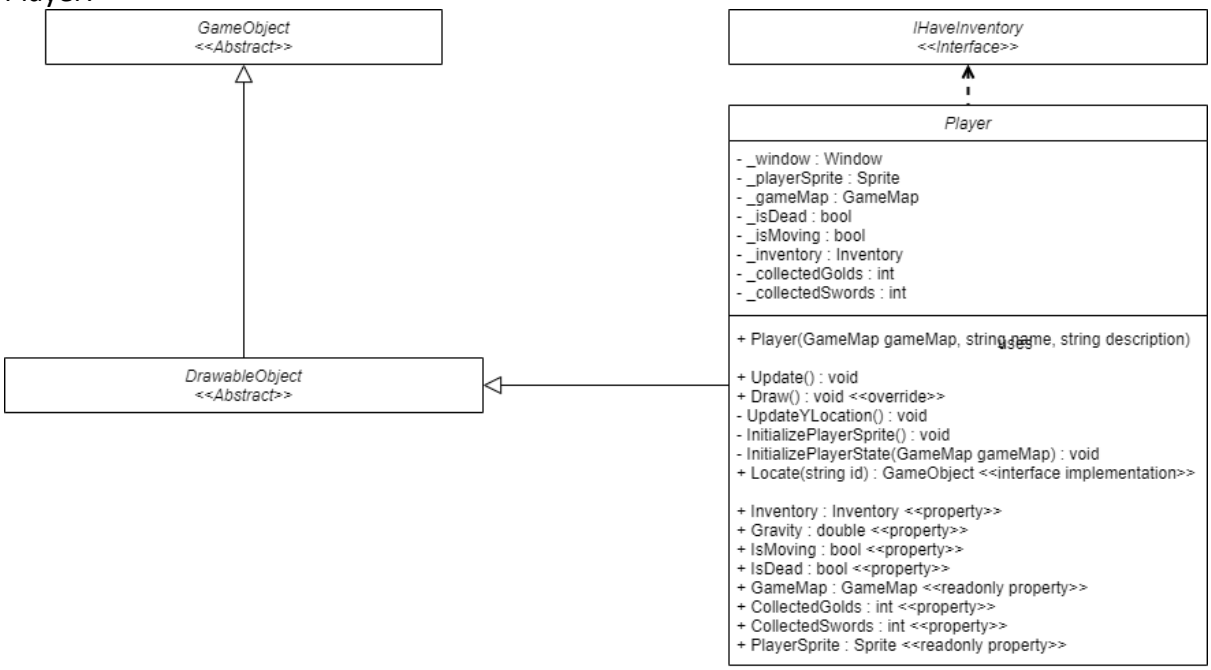
- GameMap:



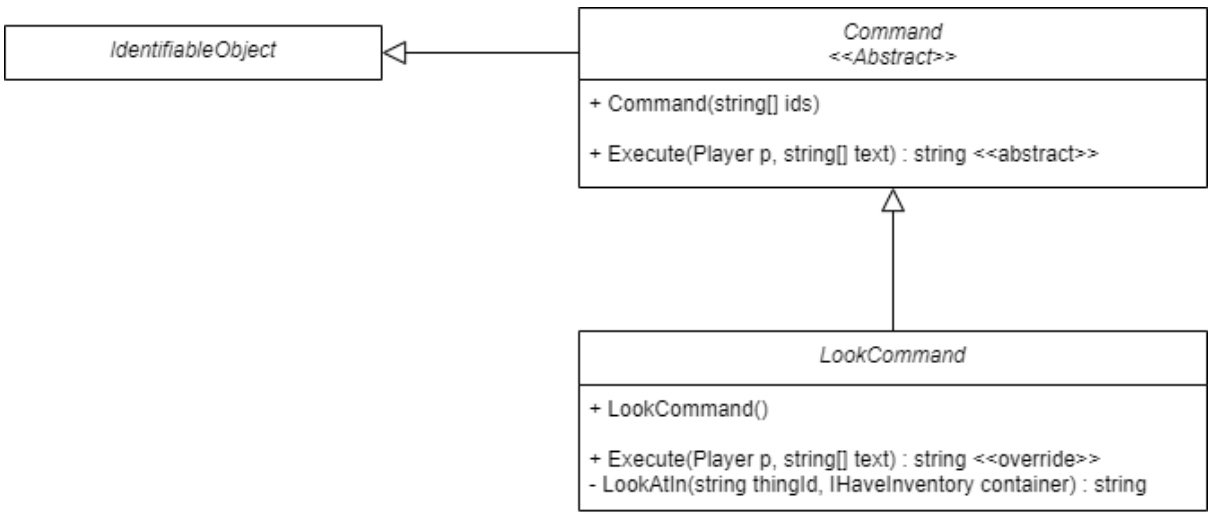
- Inventory:



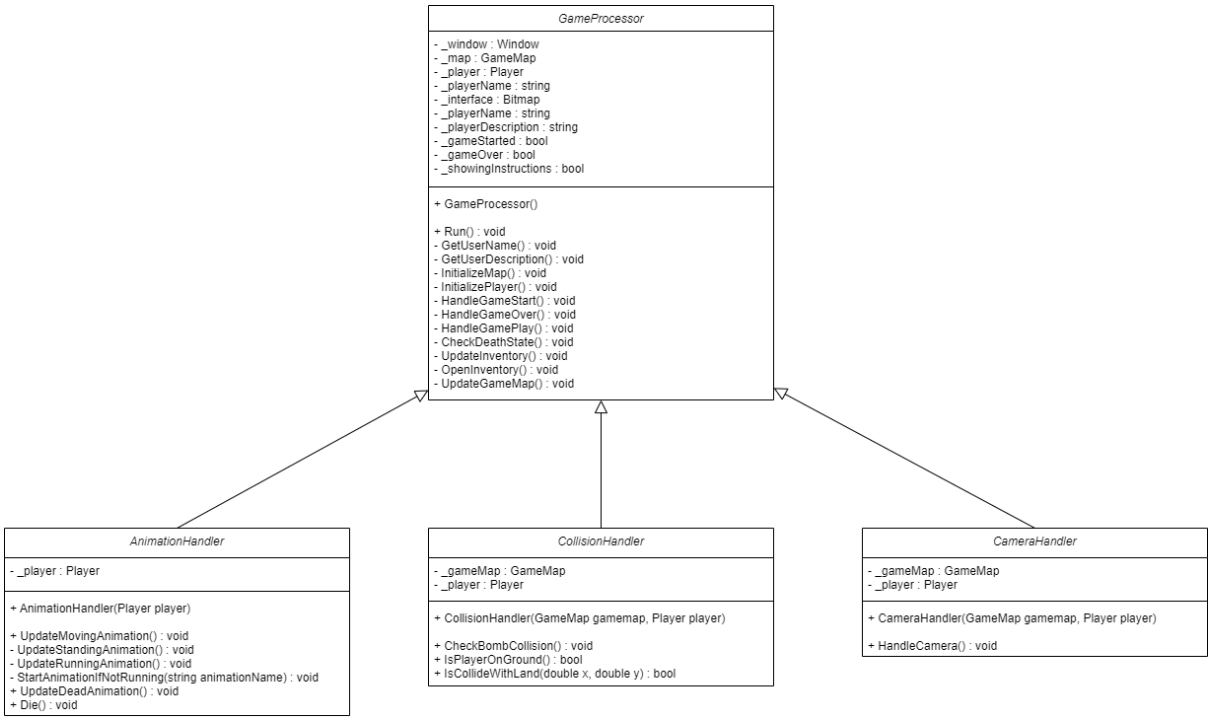
- Player:

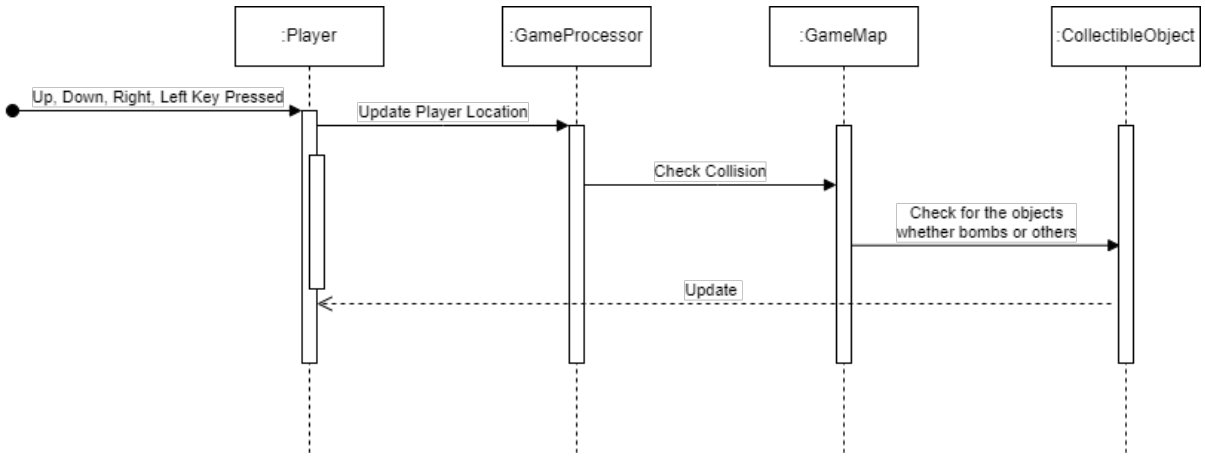


- Command:



- Game Processor:



Sequence Diagram

Here is an initial diagram showing how the game is working.