Team: Tyler Behm, John Chamberlin, Josh Killinger

Title: 2D Roguelike Game

Project Summary: For our group project, we propose to implement a 2D Roguelike game. In the following paragraphs of our proposal, we will explain what a 2D Roguelike game is, how we intend on working on it, and why it makes a good object-oriented project.

A Rougelike game is modeled after the 1980 game Rogue. A Roguelike has the player taking control of their main character. The player guides their character through a dungeon as seen through top down 2D perspective. The character may encounter enemies and other obstacles. They may use power ups or other items available in the dungeon. Most commonly sprite graphics are used to depict the character, enemies, and everything else in the game. This helps to capture the retro feel and user experience.

For our project, we will use Unity and C# to implement the game from the Unity intermediate-level tutorial exactly so we will have a working product. https://unity3d.com/learn/tutorials/projects/2d-roguelike-tutorial Then we add ghost enemies (they possess the character and forcibly move them away wasting player’s turns) and invincibility power up (10 turn immunity to enemies) objects. This will be an extension to add extra objects.

Then we refactor the game away from level progression towards screen based generated map exploration. This should be a refactor because we need to allow the player to return to the previous map square they just exited. Then we work from there time permitting.

This is a good object-oriented project because the tutorial takes advantage of many object-oriented programming paradigms. They create an abstract class MovingObject which is the class of the main character and enemy classes. This is an example of the Dependency Inversion Principle. Our extensions to the tutorial game will make use of the Open-Close Principle. Finally, our refactor towards the screen based generated map exploration will rely on good coupling to minimize the number of classes necessary to change.

Project Requirements:

No business requirements identified, as we are developing a game to be mass-marketed to a broad audience, rather than a specific client. The user field from the example requirements table is omitted. In all cases, the user is the person playing the game.

Functional requirements

ID	Requirement	Topic Area	Priority	Status
FR-01	User is presented with a procedurally-generated game map when the game starts or when a level is finished.	Gameplay	High	Existing
FR-02	Each game board consists of tiles. Some tiles can be interacted with (destroyed, consumed, etc.).	Gameplay	Med	Existing
FR-03	The character and enemy sprites are animated.	Graphical Feature	Low	Both
FR-04	The character sprite moves based on player input and will interact with objects in the destination tile if it is occupied.	Gameplay	High	Both
FR-05	The character has a limited number of turns, and the game will end if all turns are expended.	Gameplay	Med	Existing
FR-06	Players can gain additional turns by interacting with certain objects on the game board.	Gameplay	Med	Existing
FR-07	High scores are recorded in persistent data.	Additional Feature	Low	Refactor
FR-08	If the player reaches the exit tile successfully, a new level is generated.	Gameplay	Med	Existing
FR-09	Player turns are persistent throughout gameplay levels.	Gameplay	Med	Existing
FR-10	Zero-to-many enemies are generated on the map when a level starts based on the difficulty of that level.	Gameplay	Med	Both
FR-11	Enemies’ turns occur after the player’s turns.	Gameplay	Med	Existing
FR-12	Enemies can attack an adjacent player, resulting in a reduced number of turns.	Gameplay	Med	Both
FR-13	Players will not be allowed to make invalid moves.	Gameplay	Med	Refactor

Project Requirements (continued):

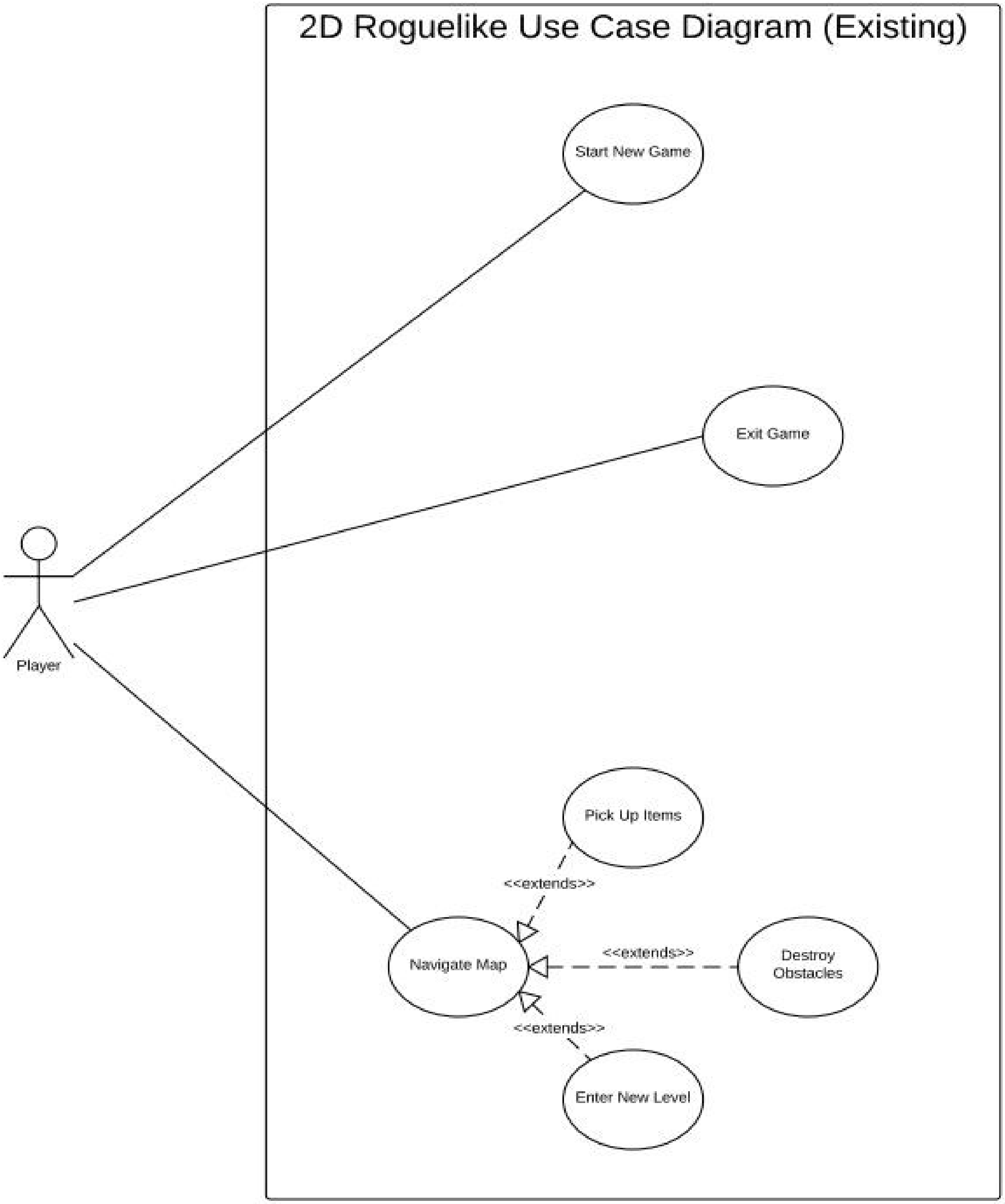
User Requirements

ID	Requirement	Topic Area	Priority	Status
US-01	Users must be presented with a menu when the game is launched, allowing them to navigate usage options.	User Interface	High	Refactor
US-02	Users must be able to start a new game.	User Interface	High	Existing
US-03	Users must be able to quit the game.	User Interface	High	Existing
US-04	Users must be able to control the volume of the game.	User Interface	Low	Refactor
US-05	Users can erase persistent data.	Additional Feature	Low	Refactor
US-06	Users must be able to access an interface menu while playing the game.	User Interface	High	Refactor
US-07	Users may navigate the game board using a basic and intuitive interface	User Interface	High	Existing
US-	Users may interact with objects on the	Gamepla	Med	Both

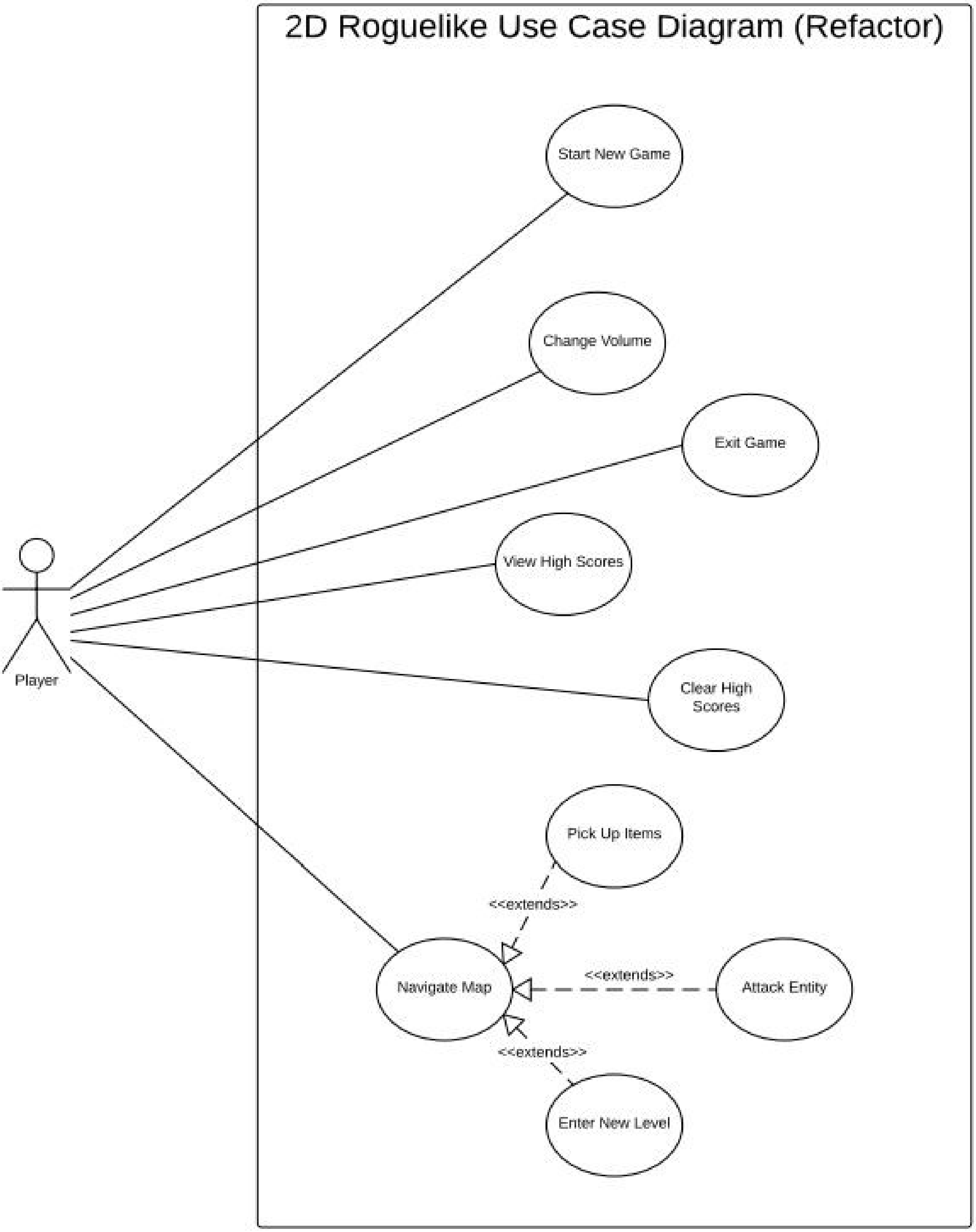
Non-Functional requirements

ID	Requirement	Topic Area	Priority	Status
NFR-01	Game functionality shall behave the same on different platforms.	Performance	High	Existing
NFR-02	The game must launch and be interactive within seven seconds.	Performance	Med	Existing
NFR-03	New levels must generate and be ready to play in five seconds.	Performance	Med	Existing
NFR-05	Enemies move in discernible patterns.	Gameplay	Low	Both
NFR-06	The map exit is always reachable.	Gameplay	Med	Existing
NFR-07	Game exits smoothly on all platforms with no errors	Performance	High	Existing

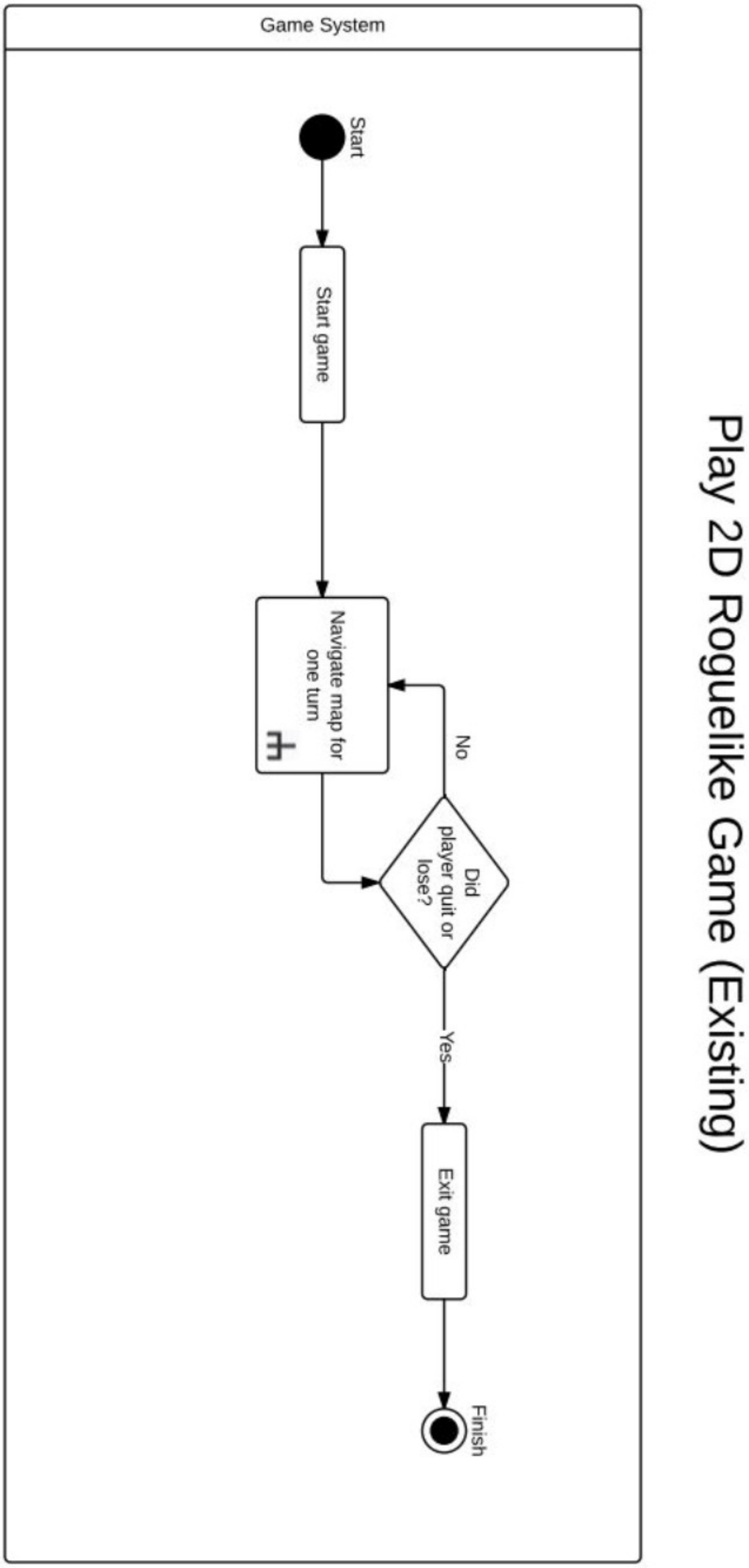
Users and Tasks (Existing State): Our design consists of one user, namely the player. She/he will be able to start a new game, exit a game, and interact with the game via map navigation, as depicted in the use case diagram below.



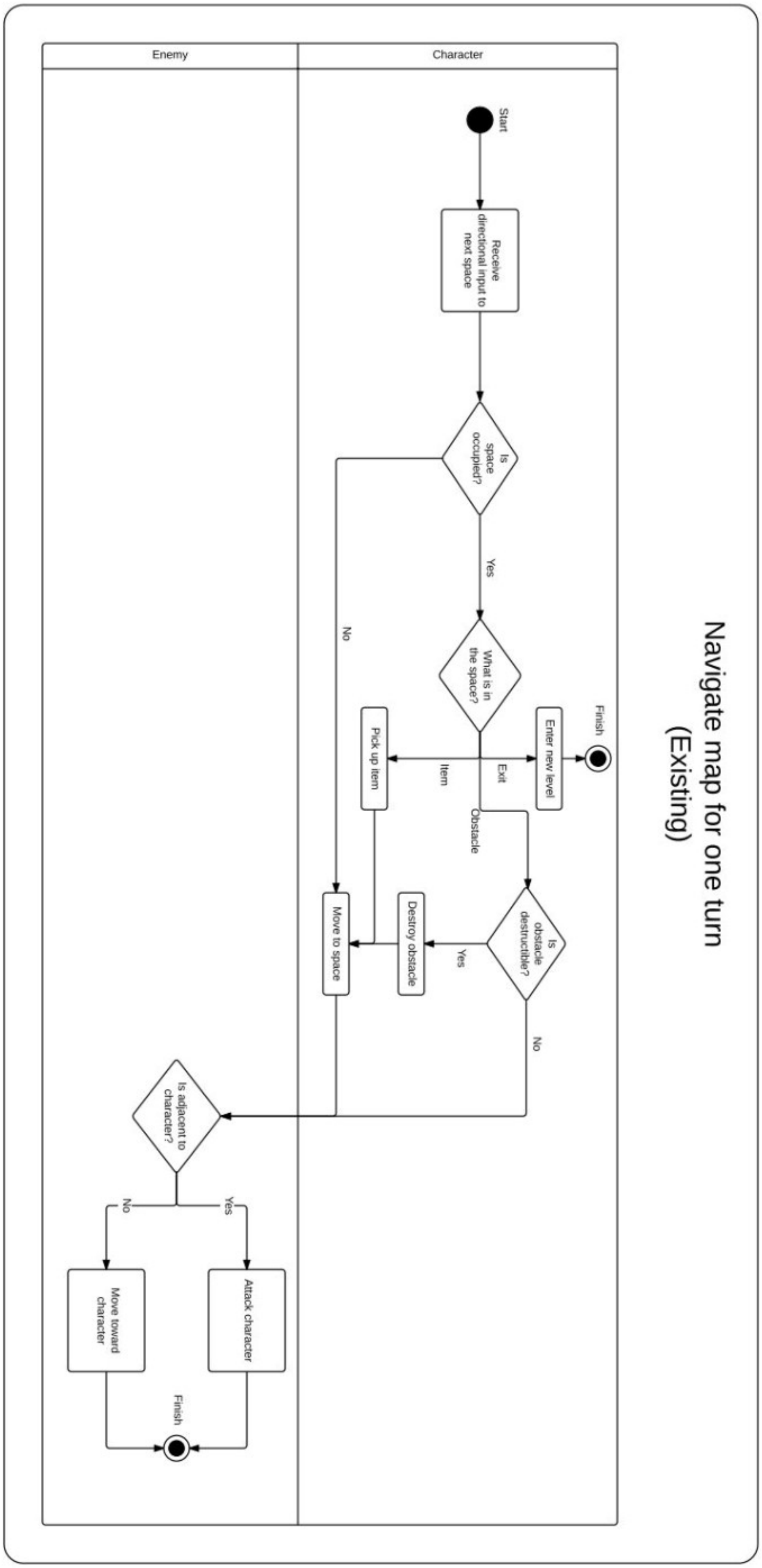
Users and Tasks (Refactor): The number and type of actors remains unchanged in our refactored design. However, we have added additional functionality which will allow the player to change the volume and view persistent high scores data.



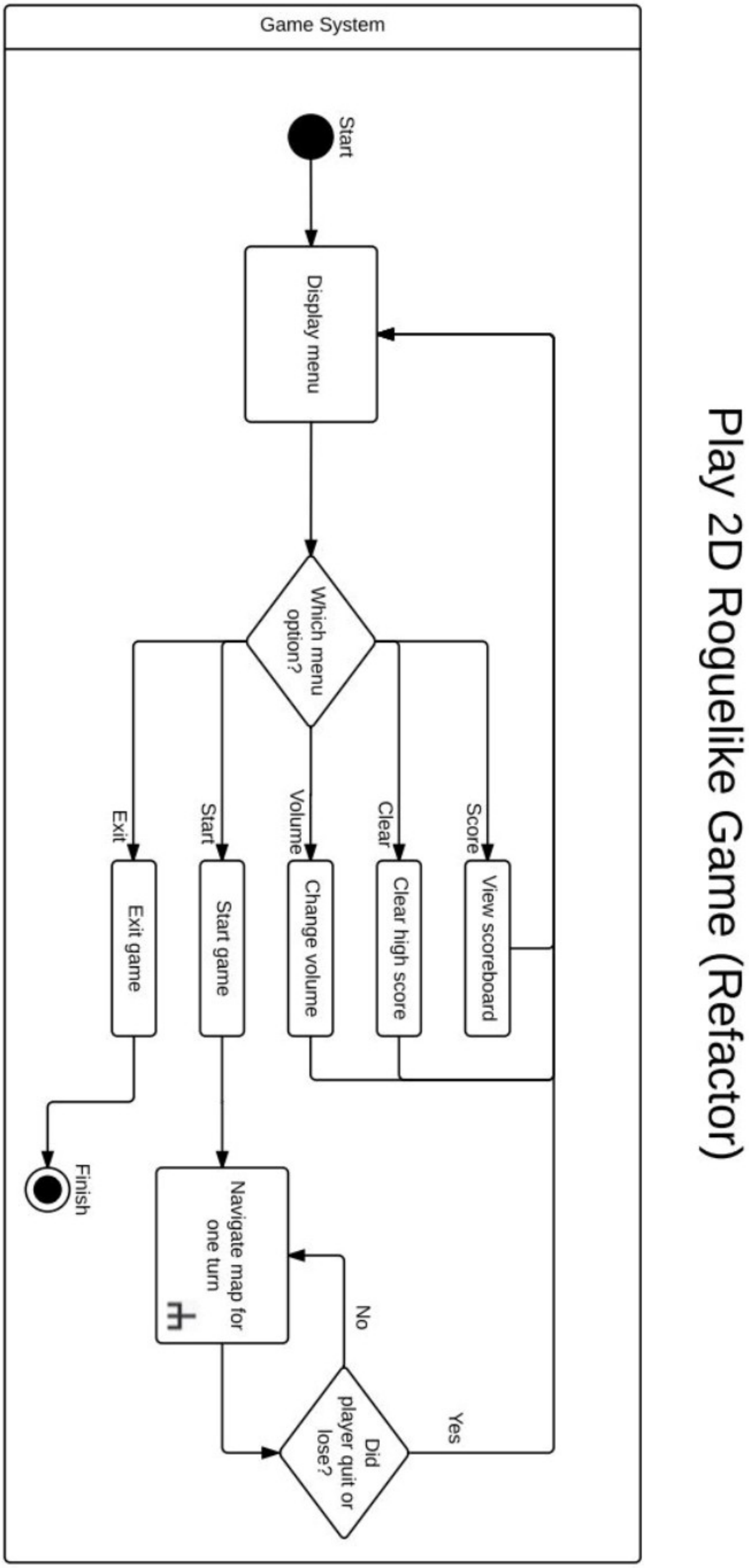
Activity Diagrams (Current State): Overall Diagram



Activity Diagrams (Current State): Navigate Map Subdiagram

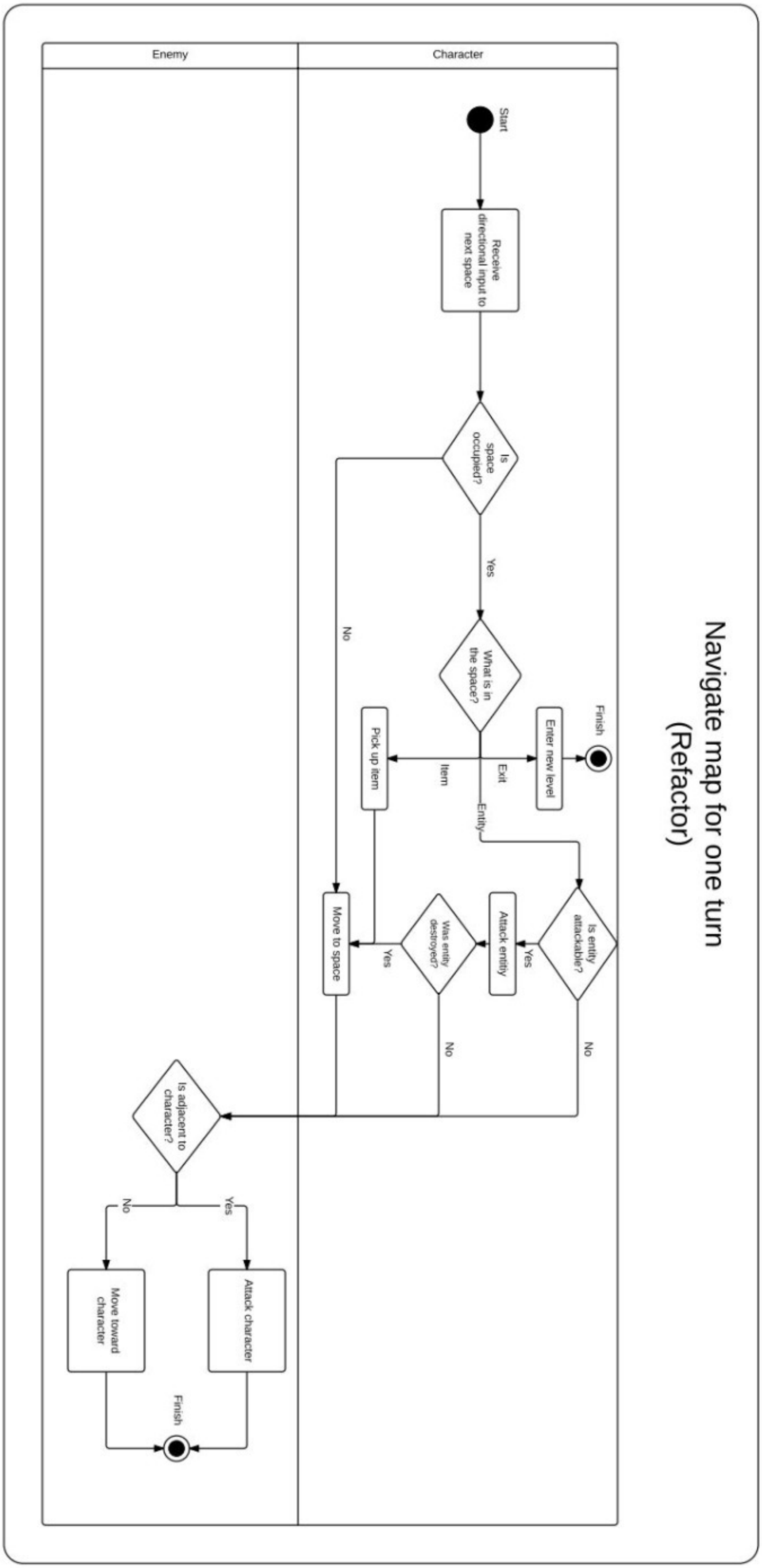


Activity Diagrams (Refactor): Overall Diagram



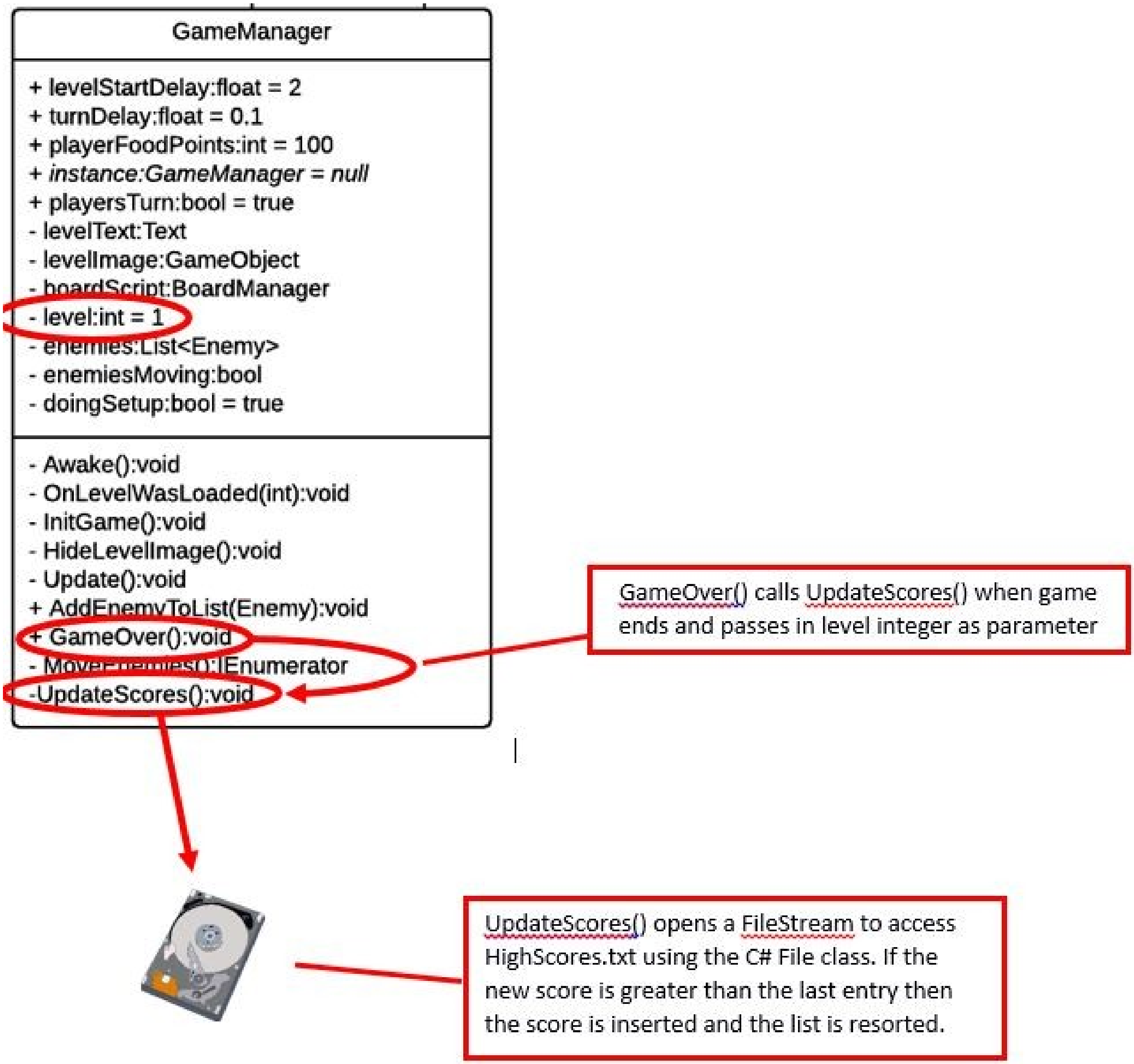
Play 2D Roguelike Game (Refactor)

Activity Diagrams (Refactor): Navigate Map Subdiagram



Data Storage (Current State): The current state does not require or use any persistent data storage.

Data Storage (Refactor): The refactored 2D Roguelike implementation will store the top ten player highest level reached and initials in a text file in the game directory. When a game ends, the GameOver() method in the GameManager object will call the updateScores() method. If the score of the game that just ended is within the top ten, then the updateScores() method will update the highscores.txt file with the new data and resort the other scores accordingly.



GameManager

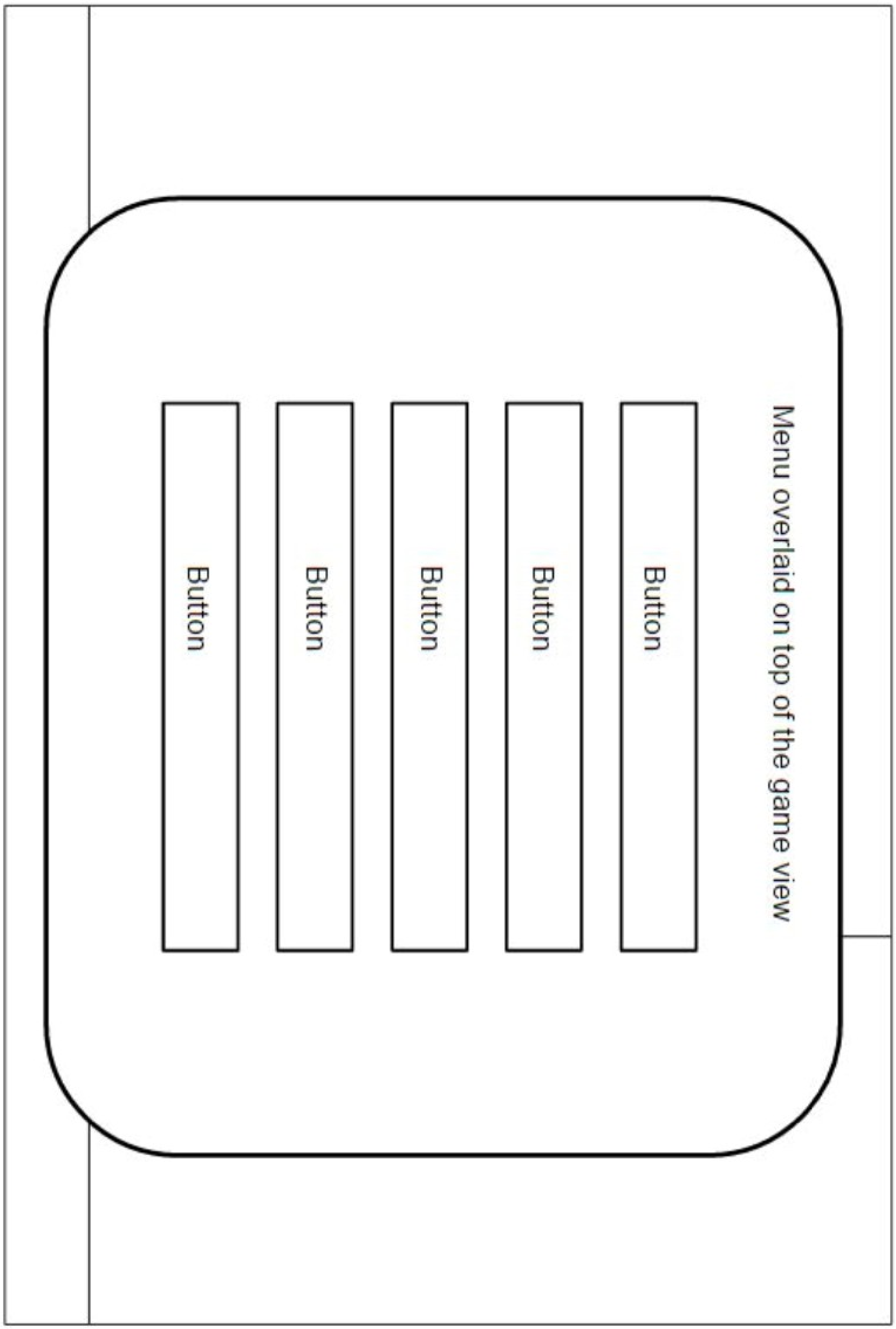
+ levelStartDelay:float = 2
+ turnDelay:float = 0.1
+ playerFoodPoints:int = 100
+ *instance:GameManager = null*
+ playersTurn:bool = true
- levelText:Text
- levelImage:GameObject
- boardScript:BoardManager
- level:int = 1
- enemies:List<Enemy>
- enemiesMoving:bool
- doingSetup:bool = true

- Awake():void
- OnLevelWasLoaded(int):void
- InitGame():void
- HideLevelImage():void
- Update():void
+ AddEnemyToList(Enemy):void
+ GameOver():void
- MoveEnemies():IEnumerator
-UpdateScores():void

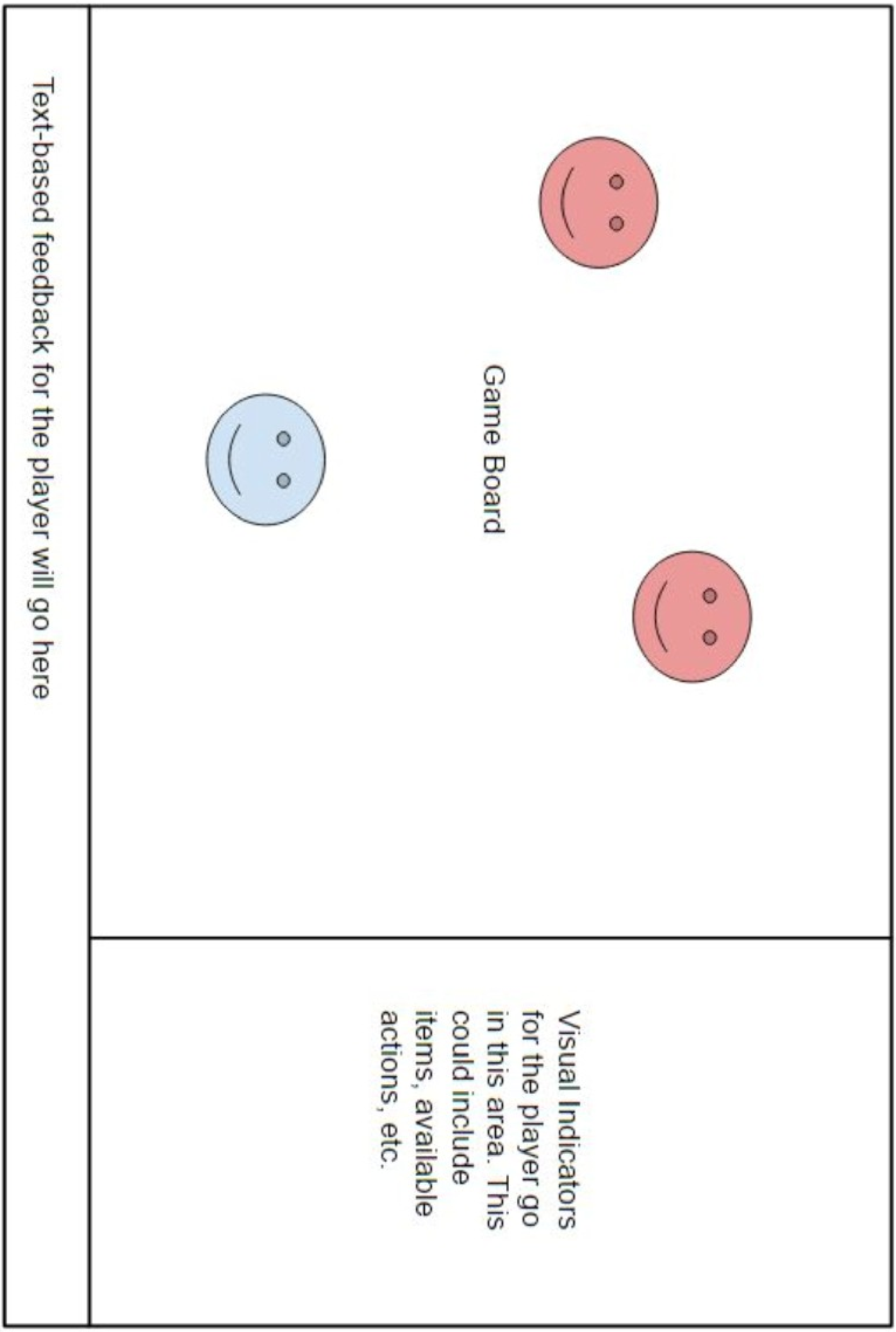
GameOver() calls UpdateScores() when game ends and passes in level integer as parameter

UpdateScores() opens a FileStream to access HighScores.txt using the C# File class. If the new score is greater than the last entry then the score is inserted and the list is resorted.

UI Mockups: Game Menu



UI Mockups: Game Board

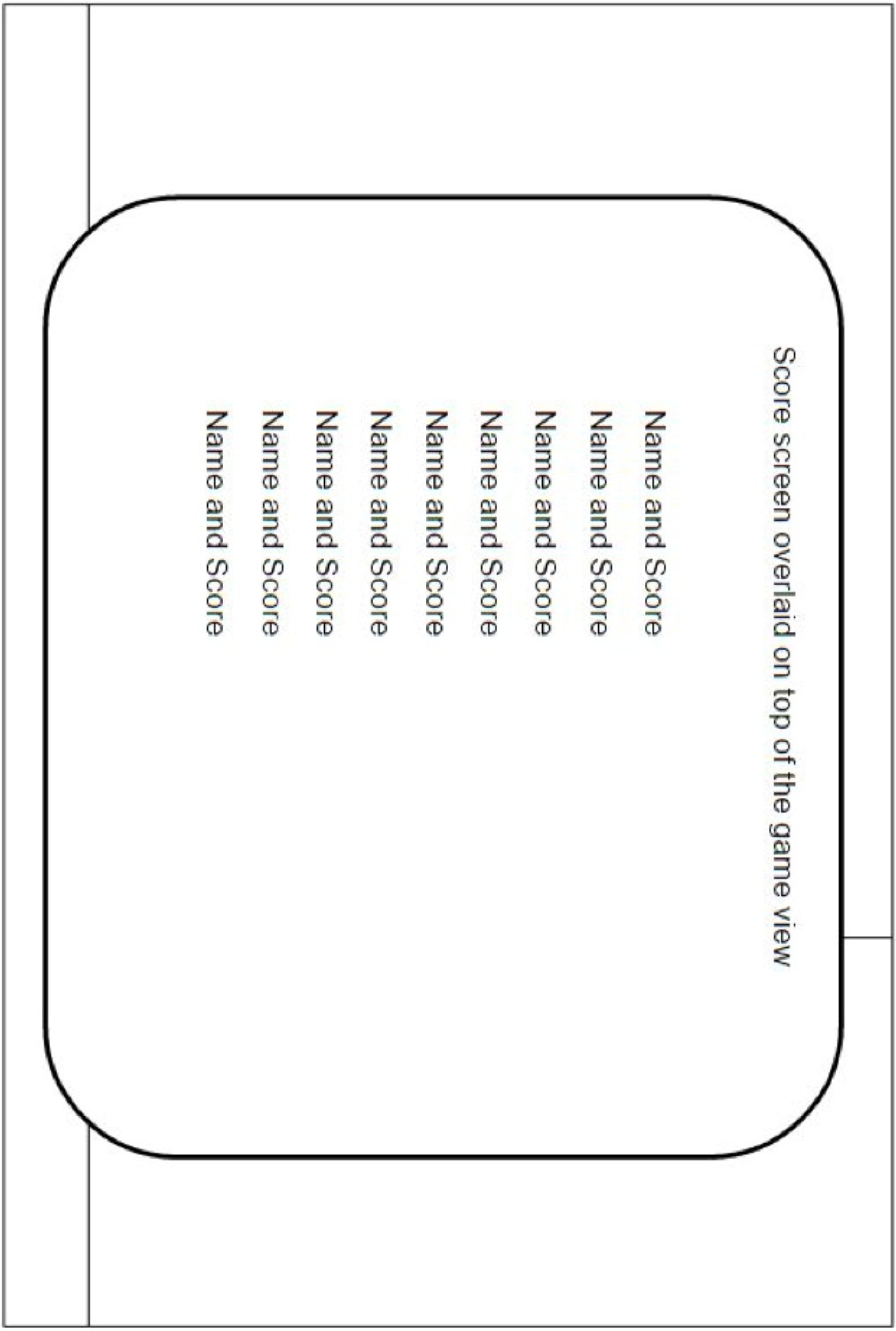


Text-based feedback for the player will go here

Game Board

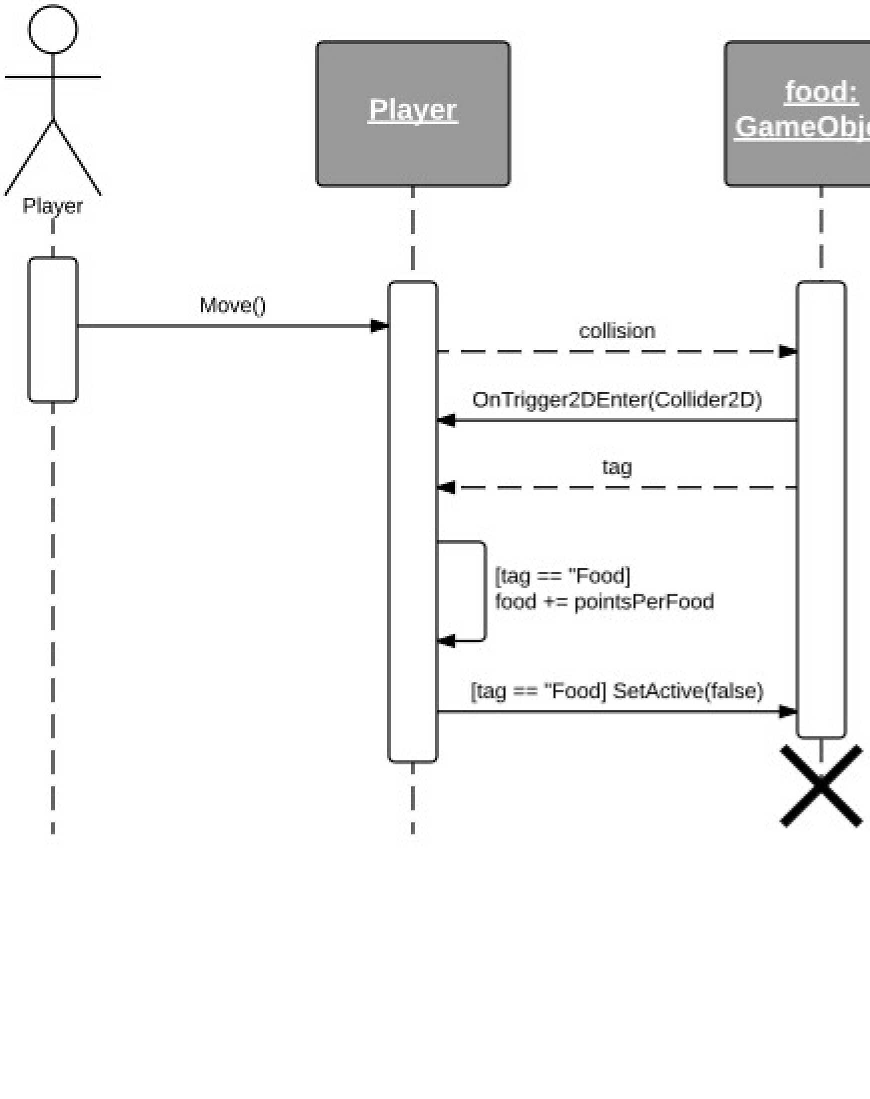
Visual Indicators for the player go in this area. This could include items, available actions, etc.

UI Mockups: High Scores Display

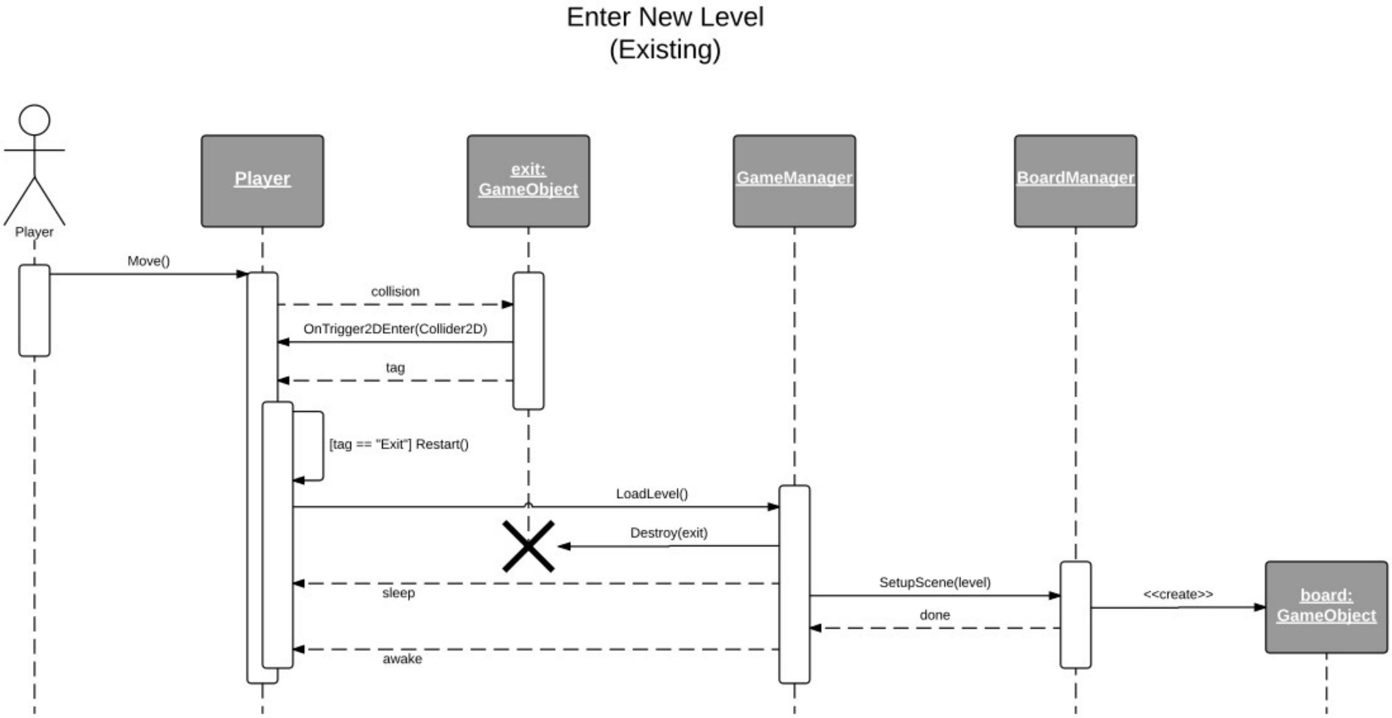


User Interactions: The “Pick Up Items” use case occurs when a player moves the player object over a food item. To know this happens, the player object needs to detect that it collided with the food object. Then player object should update its food points and destroy the food object.

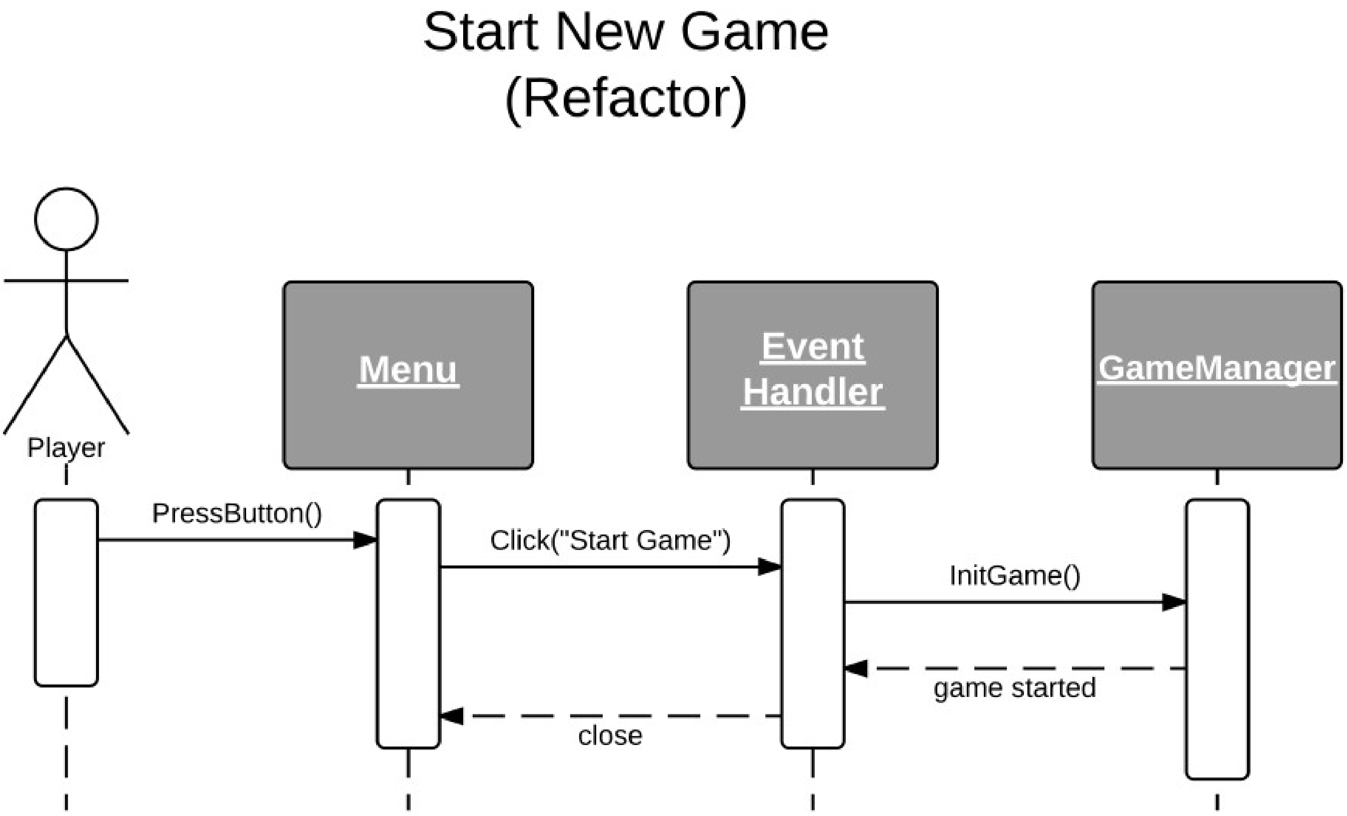
Pick Up Items
(Existing)



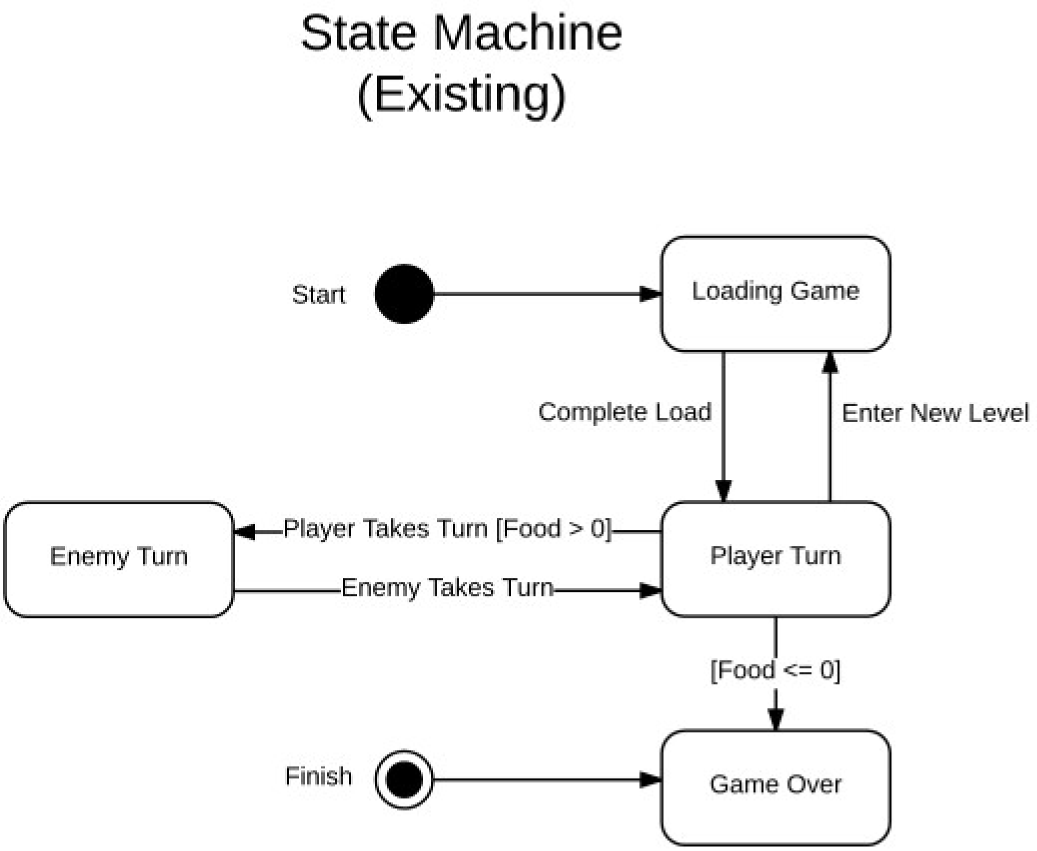
User Interactions (Continued): The “Enter New Level” use case happens when a player moves the player object over a exit object. To know this happens, the player object needs to detect that it collided with the exit object. The GameManager should then be called. The GameManager should put the player object to sleep so that it cannot move while the level is loading. Then the GameManager should destroy the old board and call the BoardManager to create the new board object. Once that's done, the BoardManager should signal the GameManager which would then wake the player to resume play.



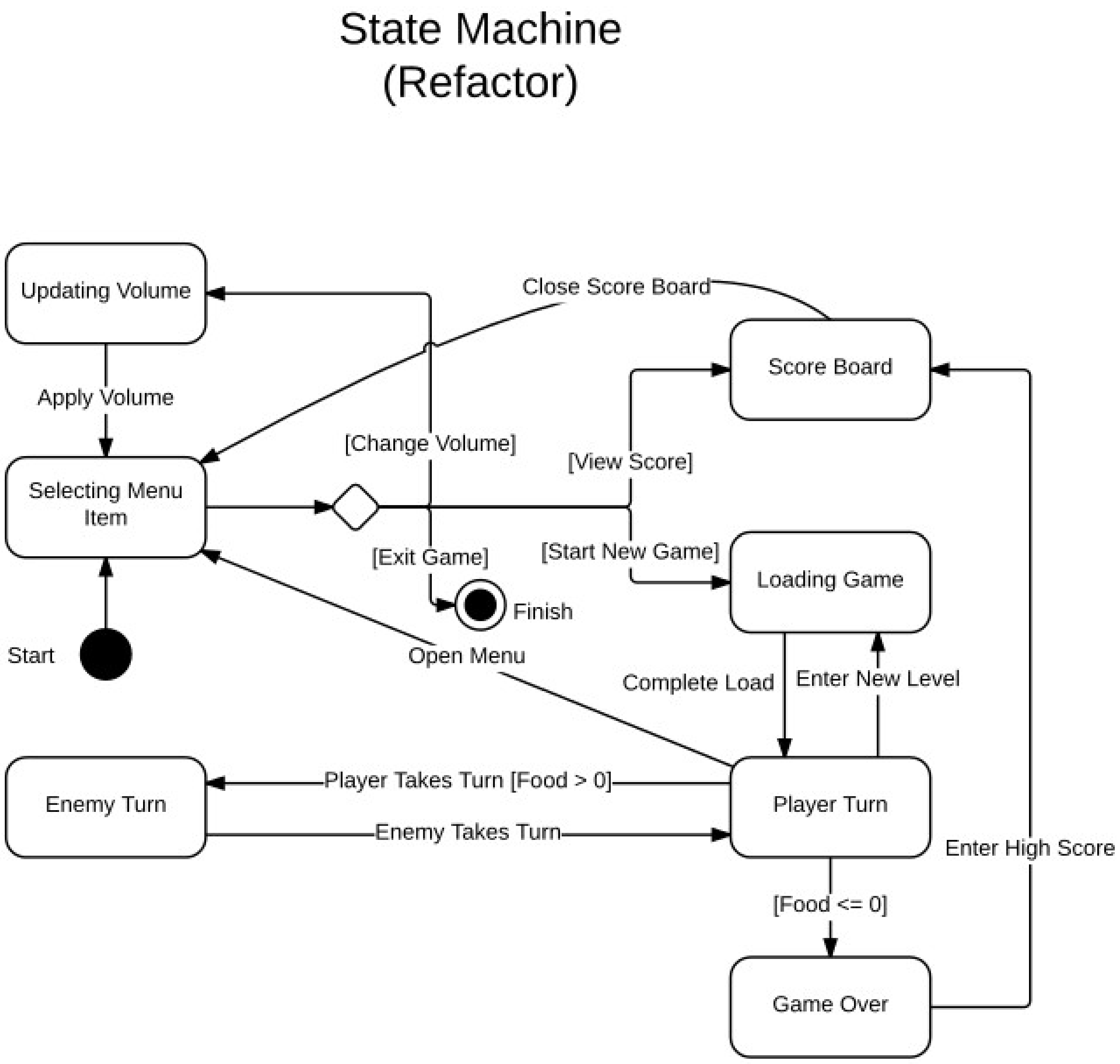
User Interactions (Continued): The refactored “Start New Game” use case will happen when the player press the “Start Game” button from within the menu that we have diagrammed in our UI mockup. The event handler will then handle
the request by calling the GameManager's initialization method to create the game. This will thensignal the menu to close so that the player can begin playing the game.



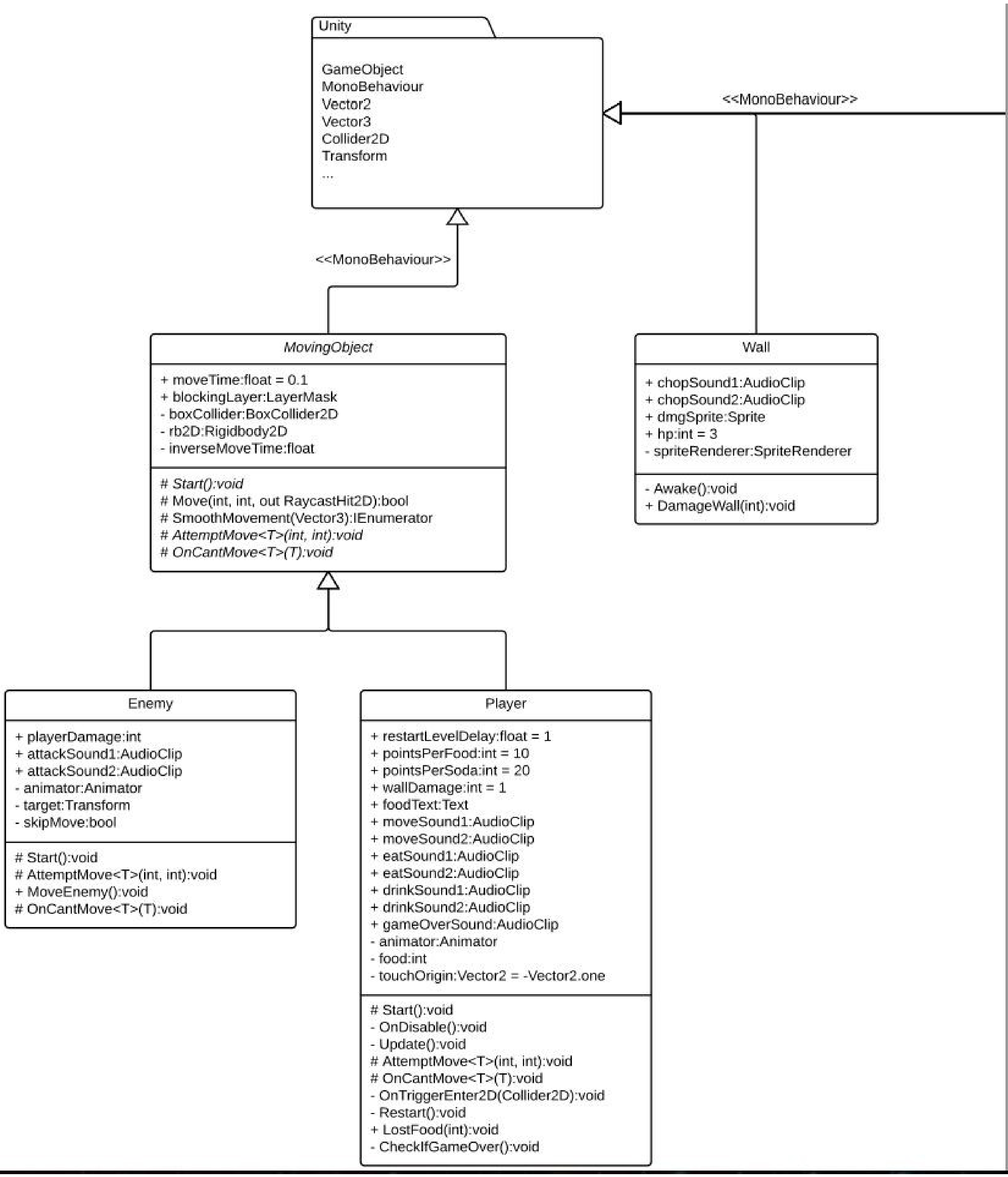
User Interactions (Continued): Current State Machine



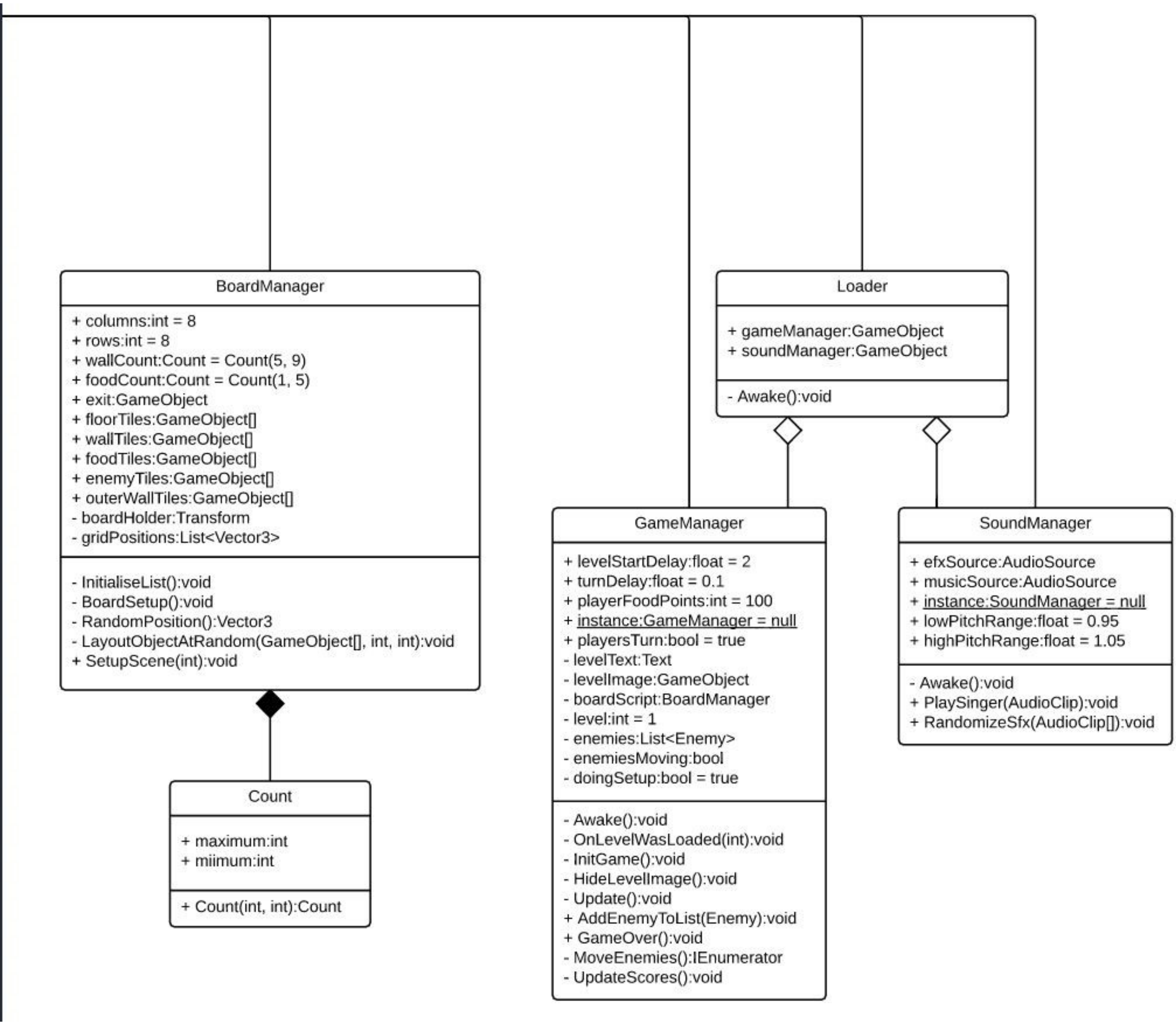
User Interactions (Continued): Refactored State Machine



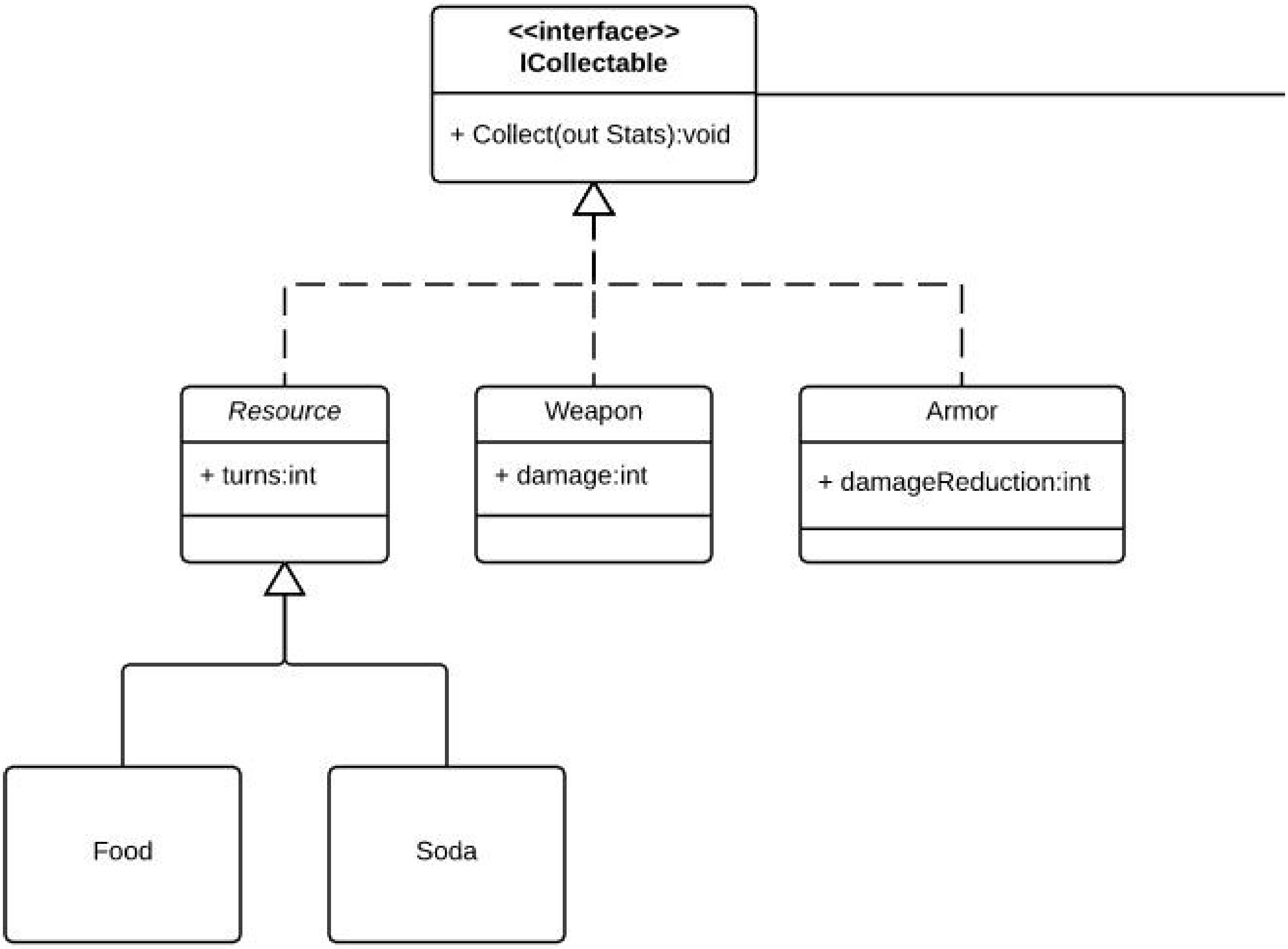
Class Diagrams: Current Left



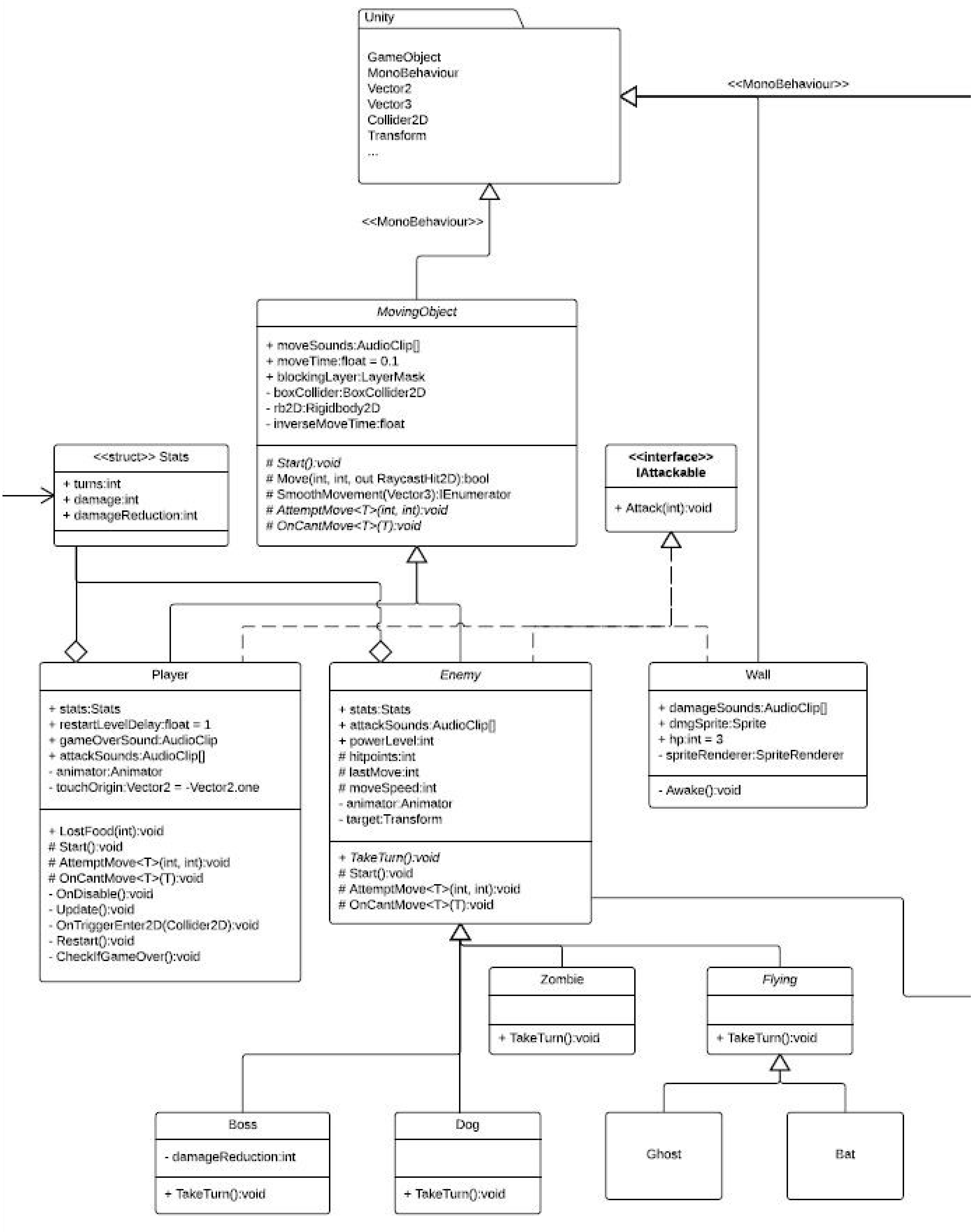
Class Diagrams (Continued): Current Right


BoardManager
+ columns:int = 8
+ rows:int = 8
+ wallCount:Count = Count(5, 9)
+ foodCount:Count = Count(1, 5)
+ exit:GameObject
+ floorTiles:GameObject[]
+ wallTiles:GameObject[]
+ foodTiles:GameObject[]
+ enemyTiles:GameObject[]
+ outerWallTiles:GameObject[]
- boardHolder:Transform
- gridPositions:List<Vector3>
- InitialiseList():void
- BoardSetup():void
- RandomPosition():Vector3
- LayoutObjectAtRandom(GameObject[], int, int):void
+ SetupScene(int):void
Count
+ maximum:int
+ miimum:int
+ Count(int, int):Count
Loader
+ gameManager:GameObject
+ soundManager:GameObject
- Awake():void
GameManager
+ levelStartDelay:float = 2
+ turnDelay:float = 0.1
+ playerFoodPoints:int = 100
+ instance:GameManager = null
+ playersTurn:bool = true
- levelText:Text
- levelImage:GameObject
- boardScript:BoardManager
- level:int = 1
- enemies:List<Enemy>
- enemiesMoving:bool
- doingSetup:bool = true
- Awake():void
- OnLevelWasLoaded(int):void
- InitGame():void
- HideLevelImage():void
- Update():void
+ AddEnemyToList(Enemy):void
+ GameOver():void
- MoveEnemies():IEnumerator
- UpdateScores():void
SoundManager
+ efxSource:AudioSource
+ musicSource:AudioSource
+ instance:SoundManager = null
+ lowPitchRange:float = 0.95
+ highPitchRange:float = 1.05
- Awake():void
+ PlaySinger(AudioClip):void
+ RandomizeSfx(AudioClip[]):void

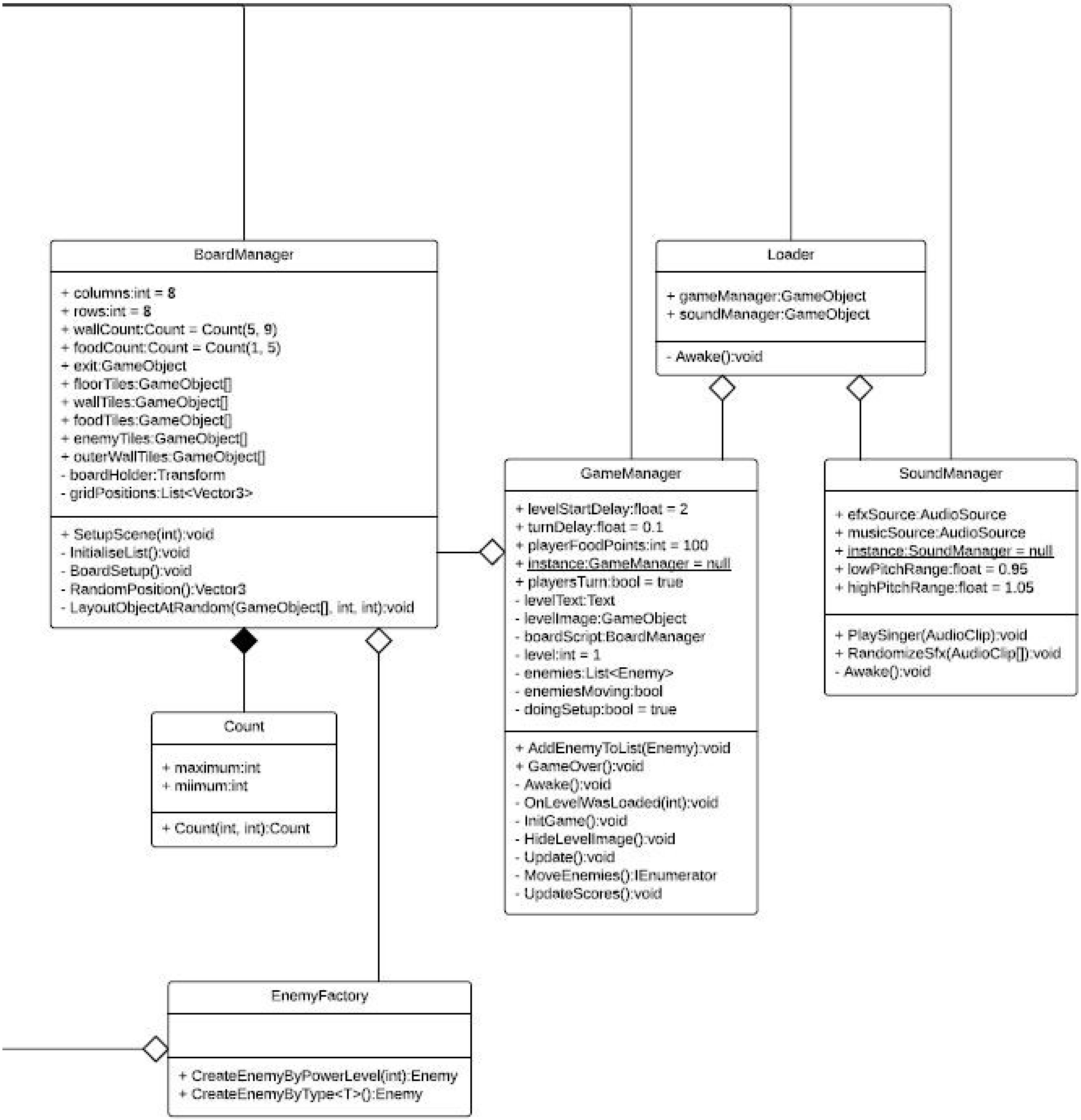
Class Diagrams (Continued): Refactor Left



Class Diagrams (Continued): Refactor Center



Class Diagrams (Continued): Refactor Right



BoardManager

+ columns:int = 8
+ rows:int = 8
+ wallCount:Count = Count(5, 9)
+ foodCount:Count = Count(1, 5)
+ exit:GameObject
+ floorTiles:GameObject[]
+ wallTiles:GameObject[]
+ foodTiles:GameObject[]
+ enemyTiles:GameObject[]
+ outerWallTiles:GameObject[]
- boardHolder:Transform
- gridPositions:List<Vector3>

+ SetupScene(int):void
- InitialiseList():void
- BoardSetup():void
- RandomPosition():Vector3
- LayoutObjectAtRandom(GameObject[], int, int):void

Loader

+ gameManager:GameObject
+ soundManager:GameObject

- Awake():void

GameManager

+ levelStartDelay:float = 2
+ turnDelay:float = 0.1
+ playerFoodPoints:int = 100
+ instance:GameManager = null
+ playersTurn:bool = true
- levelText:Text
- levelImage:GameObject
- boardScript:BoardManager
- level:int = 1
- enemies:List<Enemy>
- enemiesMoving:bool
- doingSetup:bool = true

+ AddEnemyToList(Enemy):void
+ GameOver():void
- Awake():void
- OnLevelWasLoaded(int):void
- InitGame():void
- HideLevelImage():void
- Update():void
- MoveEnemies():IEnumerator
- UpdateScores():void

SoundManager

+ efxSource:AudioSource
+ musicSource:AudioSource
+ instance:SoundManager = null
+ lowPitchRange:float = 0.95
+ highPitchRange:float = 1.05

+ PlaySinger(AudioClip):void
+ RandomizeSfx(AudioClip[]):void
- Awake():void

Count

+ maximum:int
+ miimum:int

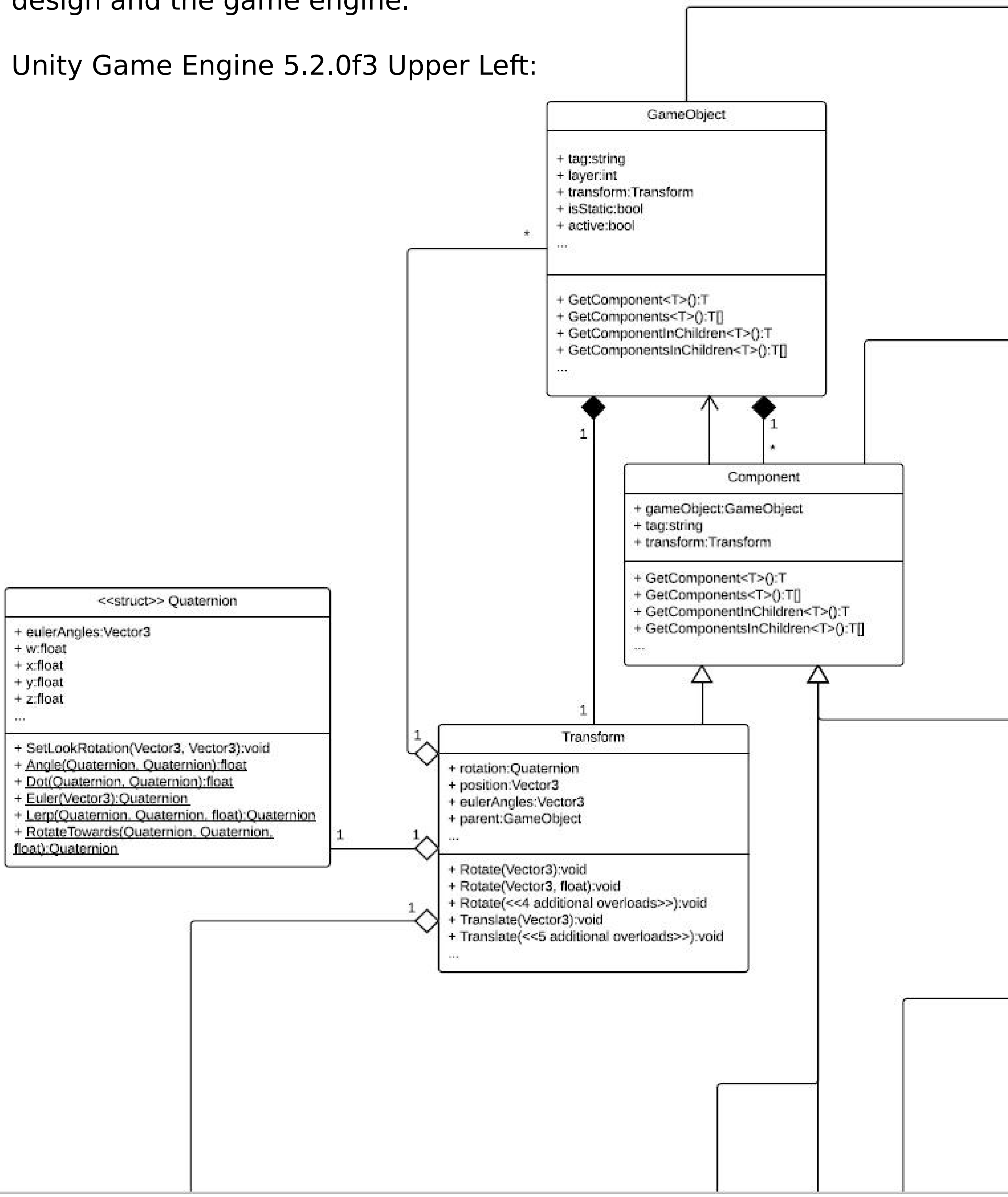
+ Count(int, int):Count

EnemyFactory

+ CreateEnemyByPowerLevel(int):Enemy
+ CreateEnemyByType<T>():Enemy

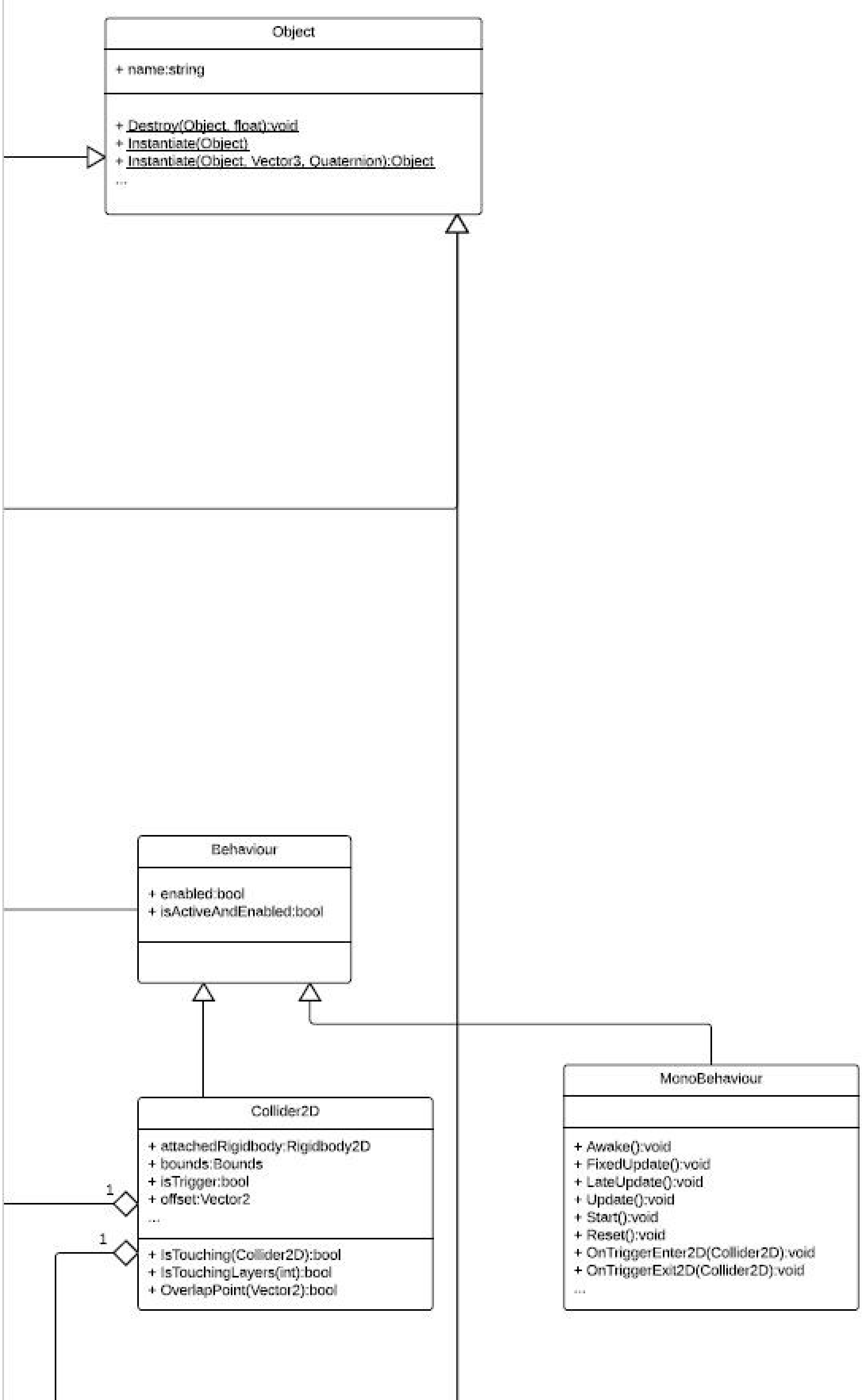
Class Diagrams (Continued): We have included the Class Diagram for the Unity Engine to illustrate the interations between our game design and the game engine.

Unity Game Engine 5.2.0f3 Upper Left:

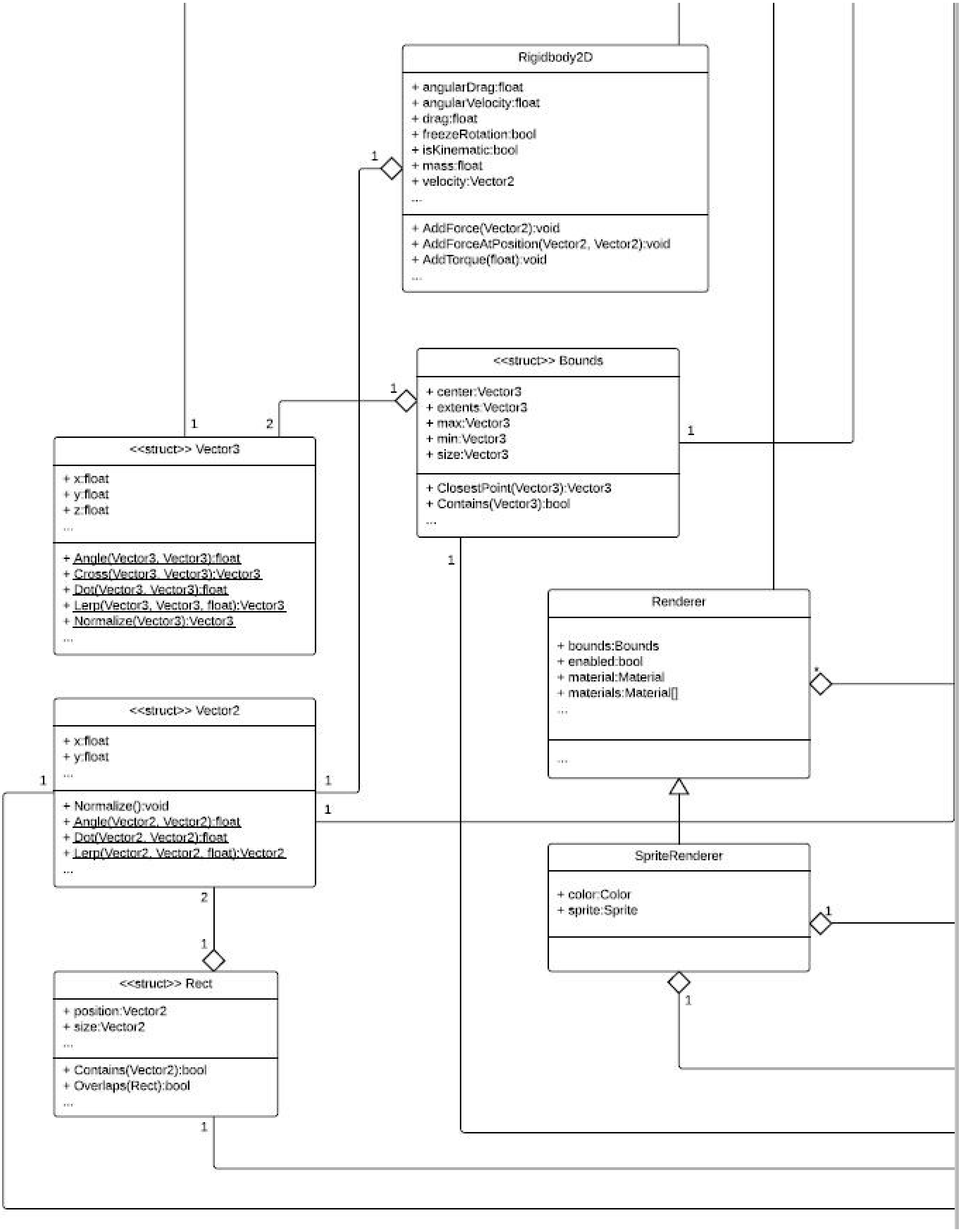


GameObject
+ tag:string
+ layer:int
+ transform:Transform
+ isStatic:bool
+ active:bool
...
+ GetComponent<T>():T
+ GetComponents<T>():T[]
+ GetComponentInChildren<T>():T
+ GetComponentsInChildren<T>():T[]
...
*
1
1
*
1
Component
+ gameObject:GameObject
+ tag:string
+ transform:Transform
+ GetComponent<T>():T
+ GetComponents<T>():T[]
+ GetComponentInChildren<T>():T
+ GetComponentsInChildren<T>():T[]
...
<<struct>> Quaternion
+ eulerAngles:Vector3
+ w:float
+ x:float
+ y:float
+ z:float
...
+ SetLookRotation(Vector3, Vector3):void
+ Angle(Quaternion, Quaternion):float
+ Dot(Quaternion, Quaternion):float
+ Euler(Vector3):Quaternion
+ Lerp(Quaternion, Quaternion, float):Quaternion
+ RotateTowards(Quaternion, Quaternion, float):Quaternion
1
1
1
1
1
Transform
+ rotation:Quaternion
+ position:Vector3
+ eulerAngles:Vector3
+ parent:GameObject
...
+ Rotate(Vector3):void
+ Rotate(Vector3, float):void
+ Rotate(<<4 additional overloads>>):void
+ Translate(Vector3):void
+ Translate(<<5 additional overloads>>):void
...

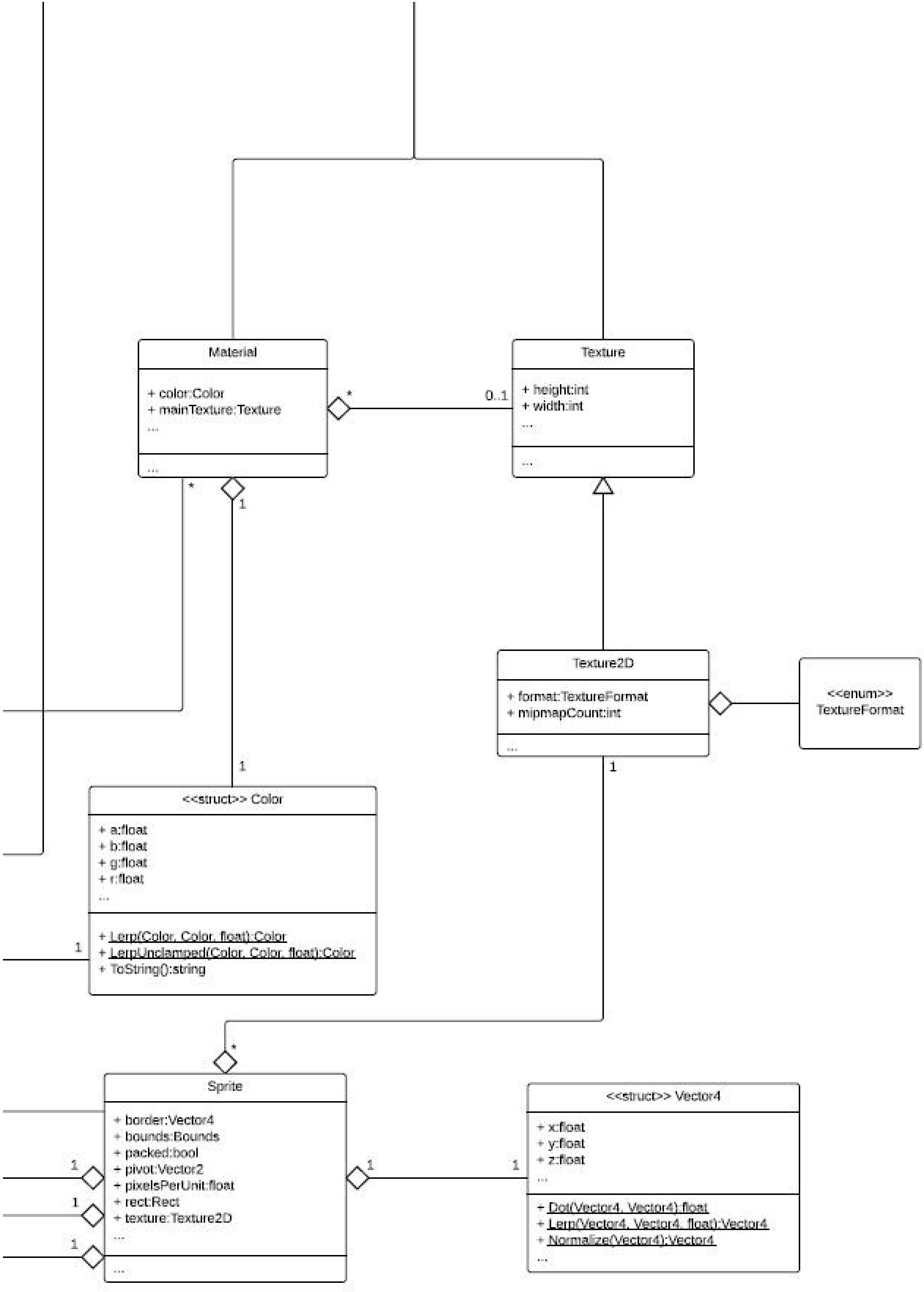
Class Diagrams (Continued): Unity Game Engine 5.2.0f3 Upper Right


Object
+ name:string
+ Destroy(Object, float):void
+ Instantiate(Object)
+ Instantiate(Object, Vector3, Quaternion):Object
...
Behaviour
+ enabled:bool
+ isActiveAndEnabled:bool
Collider2D
+ attachedRigidbody:Rigidbody2D
+ bounds:Bounds
+ isTrigger:bool
+ offset:Vector2
...
+ IsTouching(Collider2D):bool
+ IsTouchingLayers(int):bool
+ OverlapPoint(Vector2):bool
1
1
MonoBehaviour
+ Awake():void
+ FixedUpdate():void
+ LateUpdate():void
+ Update():void
+ Start():void
+ Reset():void
+ OnTriggerEnter2D(Collider2D):void
+ OnTriggerExit2D(Collider2D):void
...

Class Diagrams (Continued): Unity Game Engine 5.2.0f3 Lower Left



Class Diagrams (Continued): Unity Game Engine 5.2.0f3 Lower Right


Material
+ color:Color
+ mainTexture:Texture
...
...
*
0..1
Texture
+ height:int
+ width:int
...
...
*
1
1
Texture2D
+ format:TextureFormat
+ mipmapCount:int
...
<<enum>>
TextureFormat
1
<<struct>> Color
+ a:float
+ b:float
+ g:float
+ r:float
...
+ Lerp(Color, Color, float):Color
+ LerpUnclamped(Color, Color, float):Color
+ ToString():string
1
*
Sprite
+ border:Vector4
+ bounds:Bounds
+ packed:bool
+ pivot:Vector2
+ pixelsPerUnit:float
+ rect:Rect
+ texture:Texture2D
...
...
1
1
1
1
1
<<struct>> Vector4
+ x:float
+ y:float
+ z:float
...
+ Dot(Vector4, Vector4):float
+ Lerp(Vector4, Vector4, float):Vector4
+ Normalize(Vector4):Vector4
...