Architecture

Team 17:Scone Zone

Micheal Calaciuti
Sabrina Djebbar
Yaseen Khan
Tinetariro Muzunzandare
Josh Saunders
Zac Spooner

Abstract representation

A Turing machine structured as a 5-tuple can be used as an abstract representation of the game model.

$Tn = \langle Q, \Sigma, q0, A, \delta \rangle$ where

Q is a finite set of states q_0, q_acc, q_rej

Σ is the input alphabet

$q\_0 \in Q$

A, is the blank symbol

δ is the transition function

Using the player as a point of reference, consider the finite set of states Q. The initial or setup state is the default instance of the player and the possible interactions where all the relevant parameters are initialized. The play state will facilitate the interaction between the player and all the rest of the interactable entities. The exit state will close all the active instances of the play state and the game is terminated. Continuing with the abstract model, the input alphabet Σ will contain the expected symbols the player is expected to interact with at various game states. These include W, A, S, D and mouse_left. The transition function δ refers to a process over the input from the alphabet to decide a transition to a state. While this representation is prescriptive, it can be used to generate a concrete architectural representation of the game. This can be done by proving the computability of abstract architecture. A Universal Turing machine can accept the abstract Turing machine in order to generate a relevant computable solution. This solution is concrete architecture. The Universal Turing machine will accept the Turing machine described above and simulate it over a series to analyse its behaviour to test computability. Consider the language recognised by the Universal Turing $Ut = encode(Tn)encode(w)$ below:

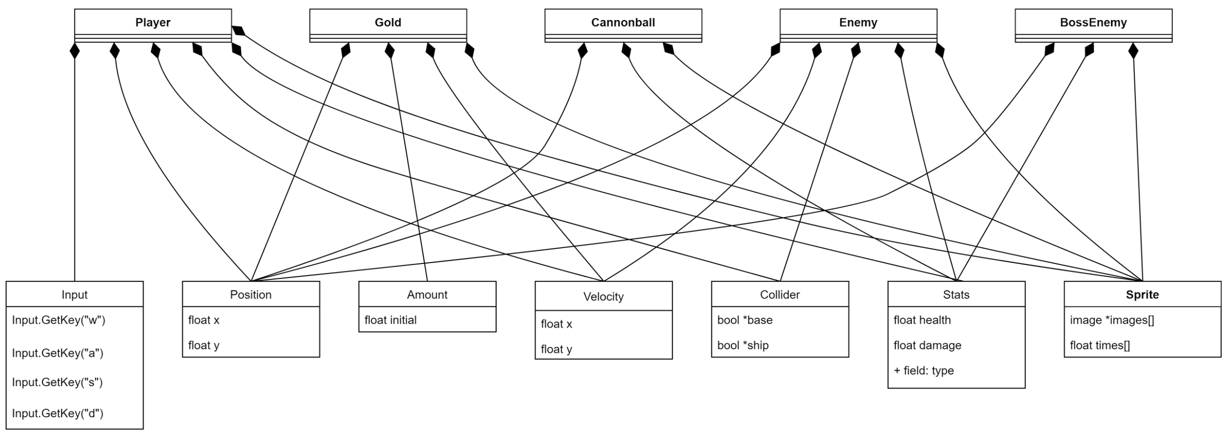
$$At = \{< Tn, w > \mid Tn\ is\ a\ turing\ machine\ that\ accepts\ w\}$$

Since Ut recognises $At$and will not halt on the input string w,Ut can be used to generate the concrete architecture since Tn is computable.

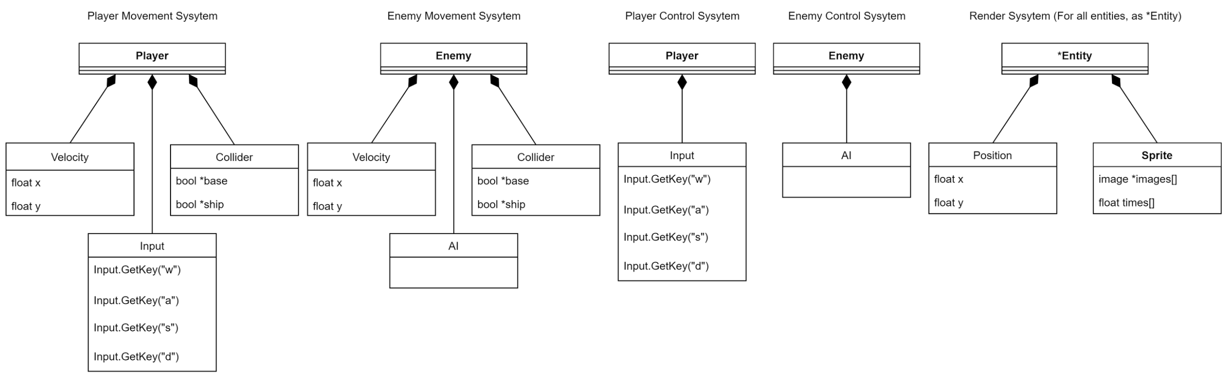
Concrete Architecture

In conjunction, the Unified Modeling Language (UML) allows the use of its nouns for structural representation such as the classes/entities and components. Also, associations are used to show the structural relationships between the entities and components in the entity-component diagram below. The entity-component diagram and the system diagram abstractly visualise the structural architecture while the sequence diagram captures the abstract behaviour. The class diagram represents the concrete architecture. The class diagram was generated from the implemented software which is based on the abstract architecture. This was achieved by using UML Lab *[1]* in Eclipse IDE.

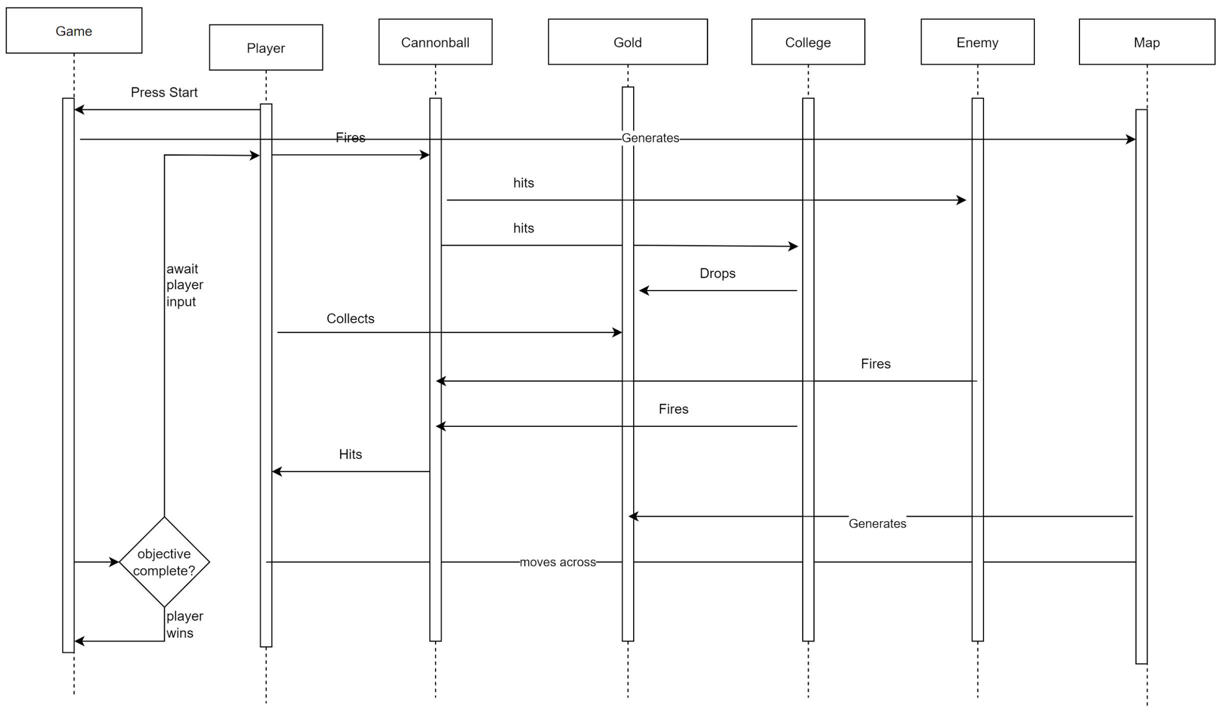
Entity-Component diagram



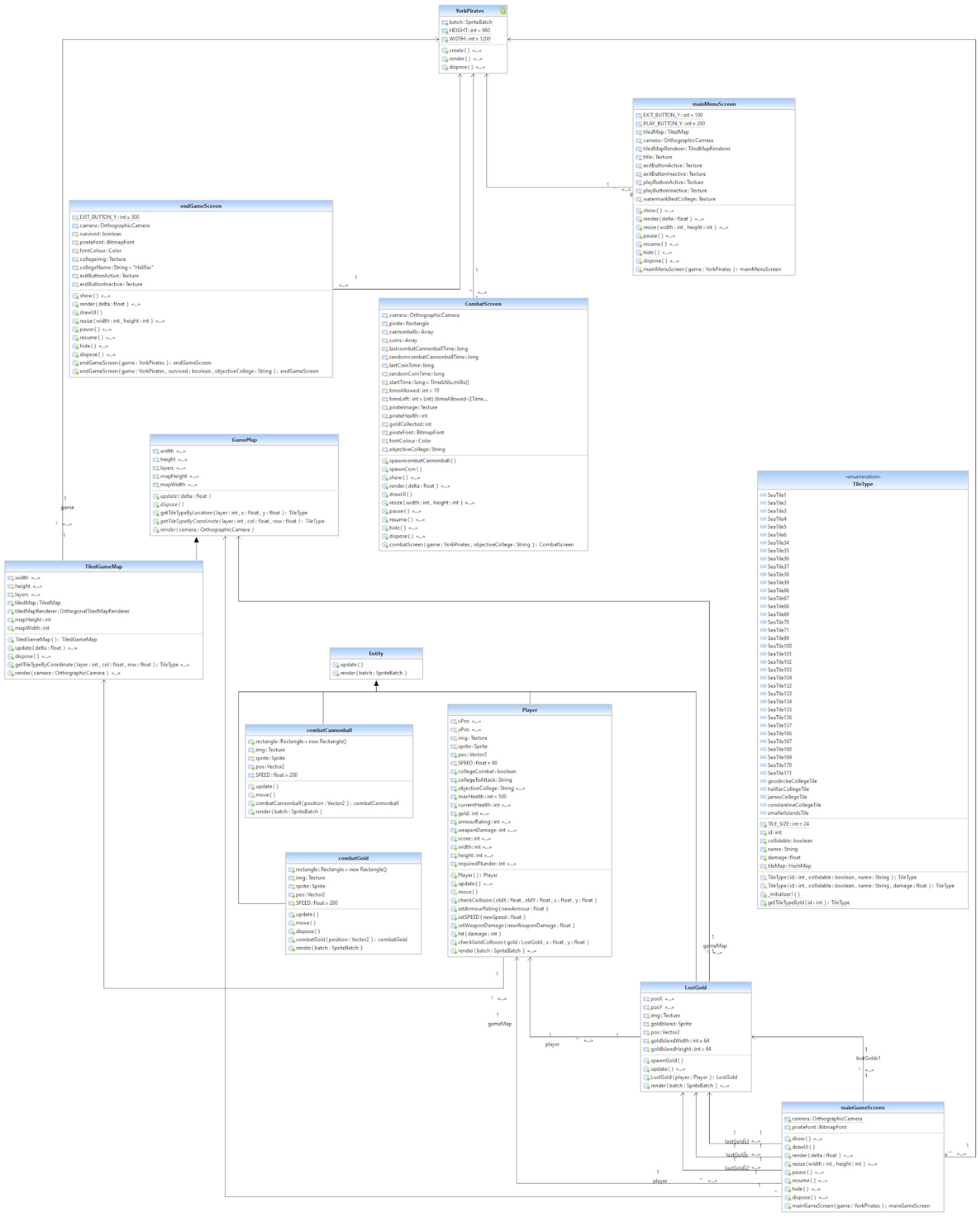
System diagram



Sequence diagram



Class diagram
(if the below diagram is too small, the full image is also viewable at https://drive.google.com/file/d/1GREIzGv5gA7bWG2bZHU4t4f_S4uwjPiY/view?usp=sharing)


YorkPirates
batch : SpriteBatch
HEIGHT : int = 960
WIDTH : int = 1200
create ( ) «...»
render ( ) «...»
dispose ( ) «...»
mainMenuScreen
EXIT_BUTTON_Y : int = 100
PLAY_BUTTON_Y : int = 200
tiledMap : TiledMap
camera : OrthographicCamera
tiledMapRenderer : TiledMapRenderer
title : Texture
exitButtonActive : Texture
exitButtonInactive : Texture
playButtonActive : Texture
playButtonInactive : Texture
watermarkBestCollege : Texture
show ( ) «...»
render (delta : float) «...»
resize (width : int , height : int) «...»
pause ( ) «...»
resume ( ) «...»
hide ( ) «...»
dispose ( ) «...»
mainMenuScreen (game : YorkPirates) : mainMenuScreen
endGameScreen
EXIT_BUTTON_Y : int = 300
camera : OrthographicCamera
survived : boolean
pirateFont : BitmapFont
fontColour : Color
collegeImg : Texture
collegeName : String = "Halifax"
exitButtonActive : Texture
exitButtonInactive : Texture
show ( ) «...»
render (delta : float) «...»
drawUI ( )
resize (width : int , height : int) «...»
pause ( ) «...»
resume ( ) «...»
hide ( ) «...»
dispose ( ) «...»
endGameScreen (game : YorkPirates) : endGameScreen
endGameScreen (game : YorkPirates , survived : boolean , objectiveCollege : String) : endGameScreen
CombatScreen
camera : OrthographicCamera
pirate : Rectangle
cannonballs : Array
coins : Array
lastcombatCannonballTime : long
randomcombatCannonballTime : long
lastCoinTime : long
randomCoinTime : long
startTime : long = TimeUtils.millis()
timeAllowed : int = 10
timeLeft : int = (int) (timeAllowed-((Time...
pirateImage : Texture
pirateHealth : int
goldCollected : int
pirateFont : BitmapFont
fontColour : Color
objectiveCollege : String
spawncombatCannonball ( )
spawnCoin ( )
show ( ) «...»
render (delta : float) «...»
drawUI ( )
resize (width : int , height : int) «...»
pause ( ) «...»
resume ( ) «...»
hide ( ) «...»
dispose ( ) «...»
combatScreen (game : YorkPirates , objectiveCollege : String) : CombatScreen
GameMap
width «...»
height «...»
layers «...»
mapHeight «...»
mapWidth «...»
update (delta : float)
dispose ( )
getTileTypeByLocation (layer : int , x : float , y : float) : TileType
getTileTypeByCoordinate (layer : int , col : float , row : float) : TileType
render (camera : OrthographicCamera)
TiledGameMap
width «...»
height «...»
layers «...»
tiledMap : TiledMap
tiledMapRenderer : OrthogonalTiledMapRenderer
mapHeight : int
mapWidth : int
TiledGameMap ( ) : TiledGameMap
update (delta : float) «...»
dispose ( ) «...»
getTileTypeByCoordinate (layer : int , col : float , row : float) : TileType «...»
render (camera : OrthographicCamera) «...»
Entity
update ( )
render (batch : SpriteBatch)
combatCannonball
rectangle : Rectangle = new Rectangle()
img : Texture
sprite : Sprite
pos : Vector2
SPEED : float = 200
update ( )
move ( )
combatCannonball (position : Vector2) : combatCannonball
render (batch : SpriteBatch)
combatGold
rectangle : Rectangle = new Rectangle()
img : Texture
sprite : Sprite
pos : Vector2
SPEED : float = 200
update ( )
move ( )
dispose ( )
combatGold (position : Vector2) : combatGold
render (batch : SpriteBatch)
Player
xPos «...»
yPos «...»
img : Texture
sprite : Sprite
pos : Vector2
SPEED : float = 90
collegeCombat : boolean
collegeToAttack : String
objectiveCollege : String «...»
maxHealth : int = 100
currentHealth : int «...»
gold : int «...»
armourRating : int «...»
weaponDamage : int «...»
score : int «...»
width : int «...»
height : int «...»
requiredPlunder : int «...»
Player ( ) : Player
update ( ) «...»
move ( )
checkCollision (oldX : float , oldY : float , x : float , y : float)
setArmourRating (newArmour : float)
setSPEED (newSpeed : float)
setWeaponDamage (newWeaponDamage : float)
hit (damage : int)
checkGoldCollision (gold : LostGold , x : float , y : float)
render (batch : SpriteBatch) «...»
«enumeration»
TileType
SeaTile1
SeaTile2
SeaTile3
SeaTile4
SeaTile5
SeaTile6
SeaTile34
SeaTile35
SeaTile36
SeaTile37
SeaTile38
SeaTile39
SeaTile66
SeaTile67
SeaTile68
SeaTile69
SeaTile70
SeaTile71
SeaTile99
SeaTile100
SeaTile101
SeaTile102
SeaTile103
SeaTile104
SeaTile132
SeaTile133
SeaTile134
SeaTile135
SeaTile136
SeaTile137
SeaTile166
SeaTile167
SeaTile168
SeaTile169
SeaTile170
SeaTile171
goodrickeCollegeTile
halifaxCollegeTile
jamesCollegeTile
constantineCollegeTile
smallerIslandsTile
TILE_SIZE : int = 24
id : int
collidable : boolean
name : String
damage : float
tileMap : HashMap
TileType (id : int , collidable : boolean , name : String) : TileType
TileType (id : int , collidable : boolean , name : String , damage : float) : TileType
_initializer1 ( )
getTileTypeById (id : int) : TileType
LostGold
posX «...»
posY «...»
img : Texture
goldIsland : Sprite
pos : Vector2
goldIslandWidth : int = 64
goldIslandHeight : int = 64
spawnGold ( )
update ( ) «...»
LostGold (player : Player) : LostGold
render (batch : SpriteBatch) «...»
mainGameScreen
camera : OrthographicCamera
pirateFont : BitmapFont
show ( ) «...»
drawUI ( )
render (delta : float) «...»
resize (width : int , height : int) «...»
pause ( ) «...»
resume ( ) «...»
hide ( ) «...»
dispose ( ) «...»
mainGameScreen (game : YorkPirates) : mainGameScreen
game
gameMap
player
lostGolds1
lostGolds3
lostGolds
lostGolds2

Systematic Justification

The abstract architecture provides a general perspective of the game's functions by showing entity-component system and sequence diagrams. The entity-component diagram shows the components of each entity that allow it to perform its expected task. This is translated into a system diagram that has the proposed sub-systems required to have a fully functioning game. Even though the sequence diagram is behavioural, it presents the expected functional flow of the game, allowing for a more robust implementation of the concrete architecture. Also, the set of finite states used to model the game as a Turing machine comprises all the game states that the player can be identified in. These include setup, play and exit state. To justify the game states, consider a player who starts the game, plays a session, and exits the game. The series of steps that the player goes through in this scenario are laid out in the sequence diagram. This helped to suggest different systems that the player would need to achieve the expected sequence plotted in the sequence diagram. These three scenarios are related to the previously stated game states, respectively.

To justify the concrete architecture, consider the implementation of the abstract architecture. By modelling and computing a Universal Turing machine that accepts the game model Turing Machine and the strings accepted by the game model Turing Machine, we can analyse the computability of the abstract architecture. After adjusting the game model Turing machine to such that it is computable, the abstract architecture is implemented into working source code. This is then used to generate the concrete architecture. The same can also be done using the entity-component, system, and sequence diagram. Since these diagrams show the expected component structure, relationships and the game flow, respectively, they give a basis for beginning implementation. During implementation, the abstract architecture develops into concrete architecture as the software is streamlined to better fit the requirement while considering workable approaches.

The software requirements and architecture are interdependent [2]. The essential requirements form the basis of abstract architecture. However, at implementation, challenges may be discovered that may require modification of the requirements. For instance, some essential requirements can conflict leading to additional requirements. Consider the architecture-requirement relations below.

Architecture-Requirement Relations

ID	Requirement	Architectural Justification
1.1	Colleges	The TileType class contains the attributes referencing the college sprites which are rendered on different screens.
1.1.1 & 1.1.1.1	Combat mechanics (capturing colleges)	The CombatScreen class has the attributes to handle all the game states for combat. The combatCannonball class facilitates the combat AI.
1.1.2	Different college themes	The TileType class contains college sprites with different fictional themes for each college.
1.1.2.1	Mega boss battle, combining all mechanics	This is not implemented yet
1.1.3	Interactables	For instance, the mainMenuScreen class handles the player's initial interaction after setup
1.2 & 1.2.1	Movement of player on the world map and Consistent movement speed between diagonal and orthogonal	The Player class contains attributes such as sprite:Sprite and SPEED:float to show the player sprite moving at a certain speed on the screen.
1.3	Functioning menu system	The mainMenuscreen class facilitates the menu functions.
1.3.1	Pauses game	The mainGameScreen class has the pause() method however this is partially implemented hence nonfunctional.
1.3.3	Settings	This is not yet implemented
1.3.3.1	Accessibility settings (Risk: 10)	This is not yet implemented
1.4	Assets for visual design	The TileType class references the game assets stored on a tile sheet.
1.5	Randomised objectives to complete	Objectives are actively fixed during the setup state.
1.6	Scoring system	The LostGold class updates the players remaining gold based on the player's interactions however, this is not independent.
1.6.1 & 1.6.2	Plunder (ie. gold) system (Gotten from completing objectives)	The CombatScreen, combatCannonball, combatGold and LostGold classes facilitate the plunder earning and recording system.
1.7	Sound (Music & SFX)	The mainGameScreen and CombatScreen class play music during the play state however, SFX is not yet implemented.

Bibliography

[1] UML Lab Modeling IDE 2010
https://marketplace.eclipse.org/content/uml-lab-modeling-ide
[2] Charlie Alfred’s Weblog Requirements vs Architecture 2008
https://charliealfred.wordpress.com/requirements-vs-architecture/