TANKMAZE

Solve the Maze!

Credits:

Ahmed Magdy Ibrahim 3003

Nader Sherif Al Sharkawy 3001

Rana Mohamed Azmy 3086

Tasneem El Sayed Alam 3055

Basma Kadry Osman 3052

Objective:

The Objective of the game is to get out of the maze without getting hit by a bomb, or attacked by an enemy, collecting gold and ammo on your way. You should reach the enemy base safely, shoot the enemy base and get ready for the next level.

Description of the Design:

The program starts by generating a randomized maze. Then, the factory design pattern is utilized to build the models onto the view. Each model can then be observed independently since an observer is implemented in each. The models are then kept in an object pool, through which every model will later be available for retrieval. A set containing all the observers that have been generated is stored in the mediator. The AI controlling the enemy tank is then activated on its respective thread and reserved timer. The player controller then takes control of handling both, the player's motion and collision with the maze components. When a bullet is fired, it is handled on an independent thread, and on collision with a destroyable maze element, the element is destroyed. A camera controller that has been designed specifically for the maze is used to facilitate the player's motion through it; displaying the appropriate area of it based on the player's coordinates within.

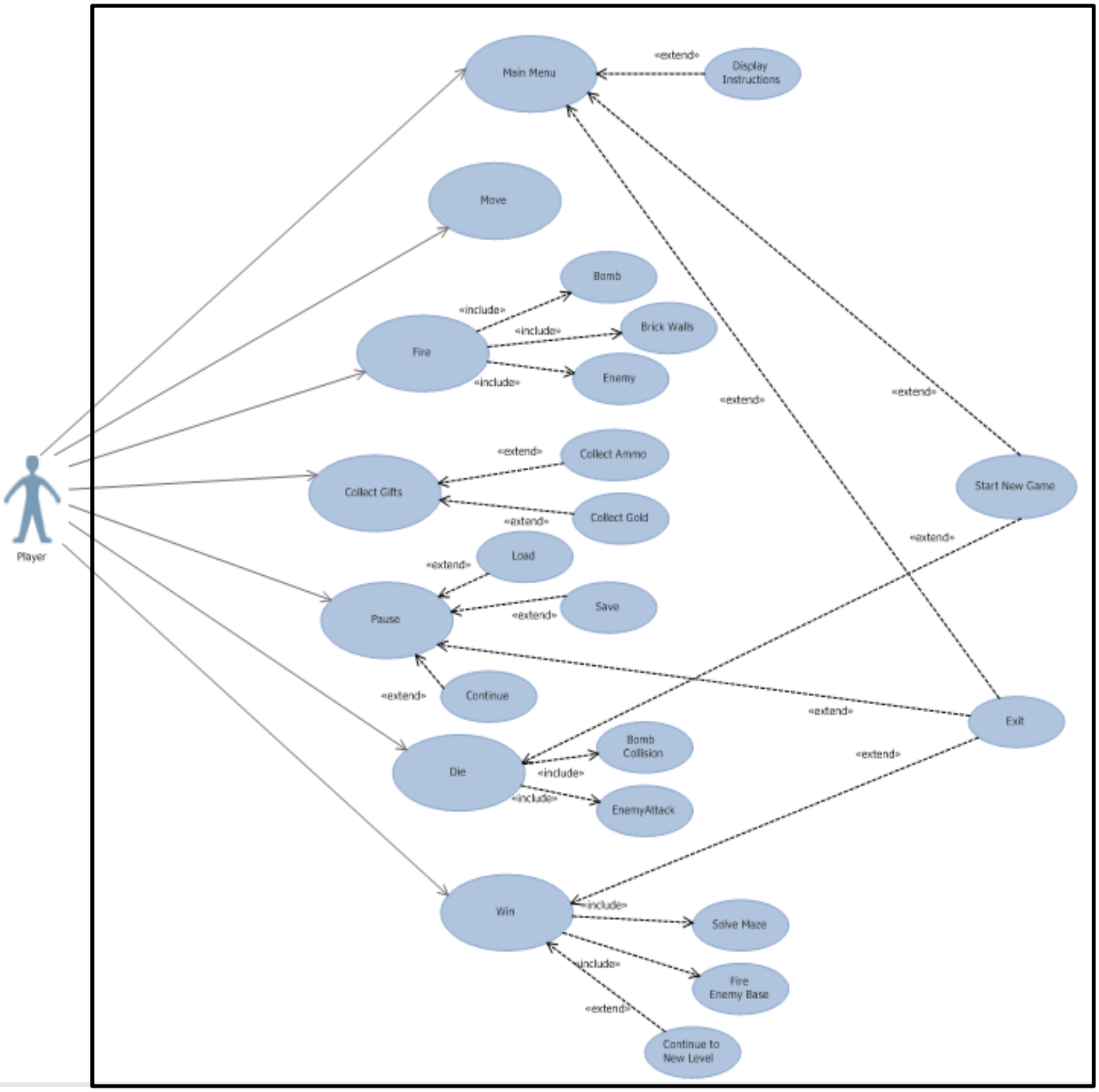
A singleton is used to make sure that one, and only one, instance of the player and the base is created per game level. When an object is destroyed, its internal state is set to nonexistent and is then removed from the playground.

Code Design:

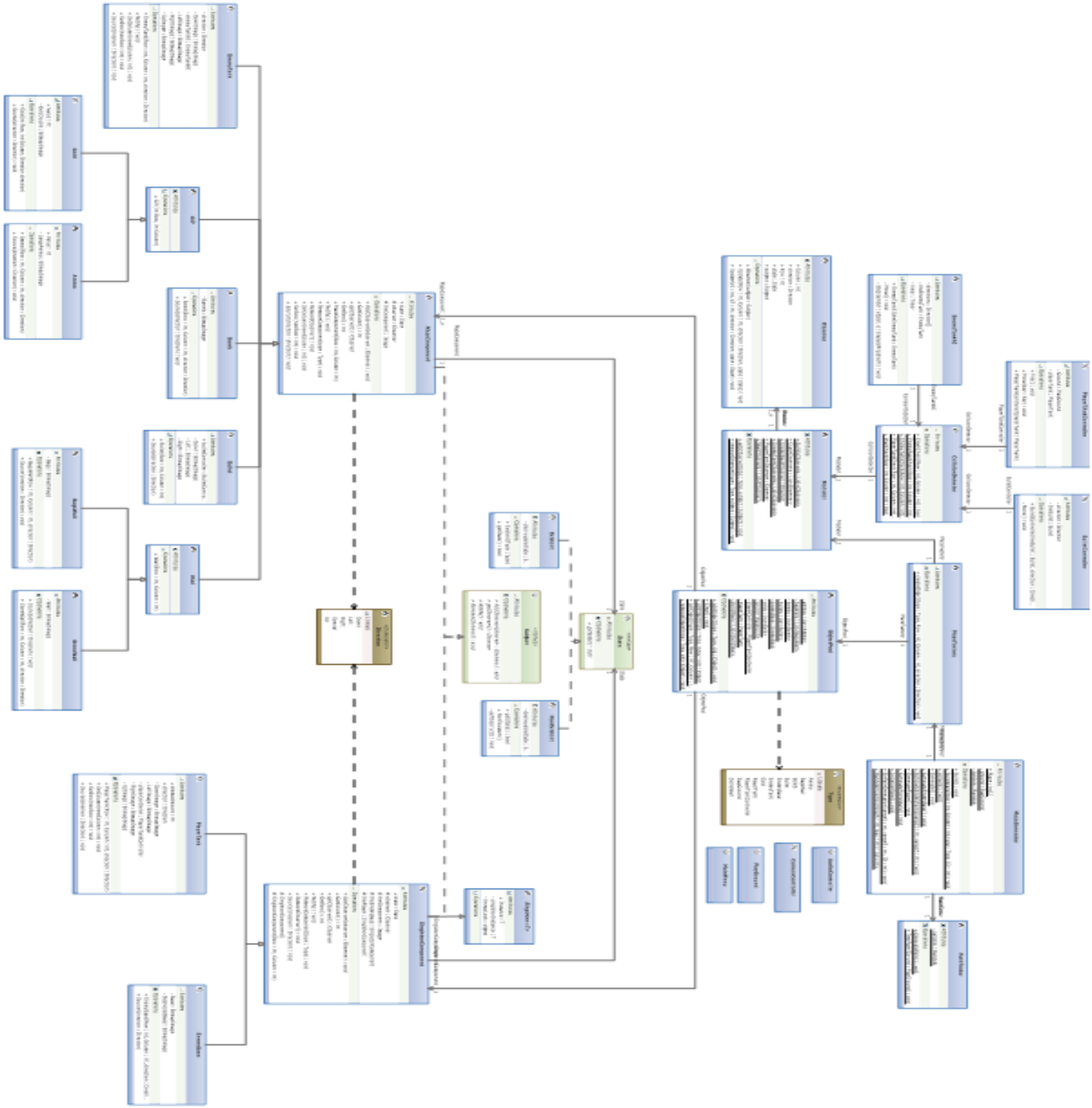
- Language: C#
- Design Patterns Used:
  - Singleton Design Pattern
  - State Design Pattern
  - Observer Design Pattern
  - Factory Design Patter
  - Object Pool Design Pattern
  - Model View Controller Design Pattern
- The Models are:
  - Player Tank, Enemy Tanks, Enemy Base, Gifts, Bombs, Walls and Bullets.
- The Controllers are:
  - Player Tank, Enemy Tank, Audio, Camera, Bullet Controllers, Maze and Path Generators, Collision Detector.
- The View is the Main Menu and the Play Ground.

Now let's take a look at the UML...

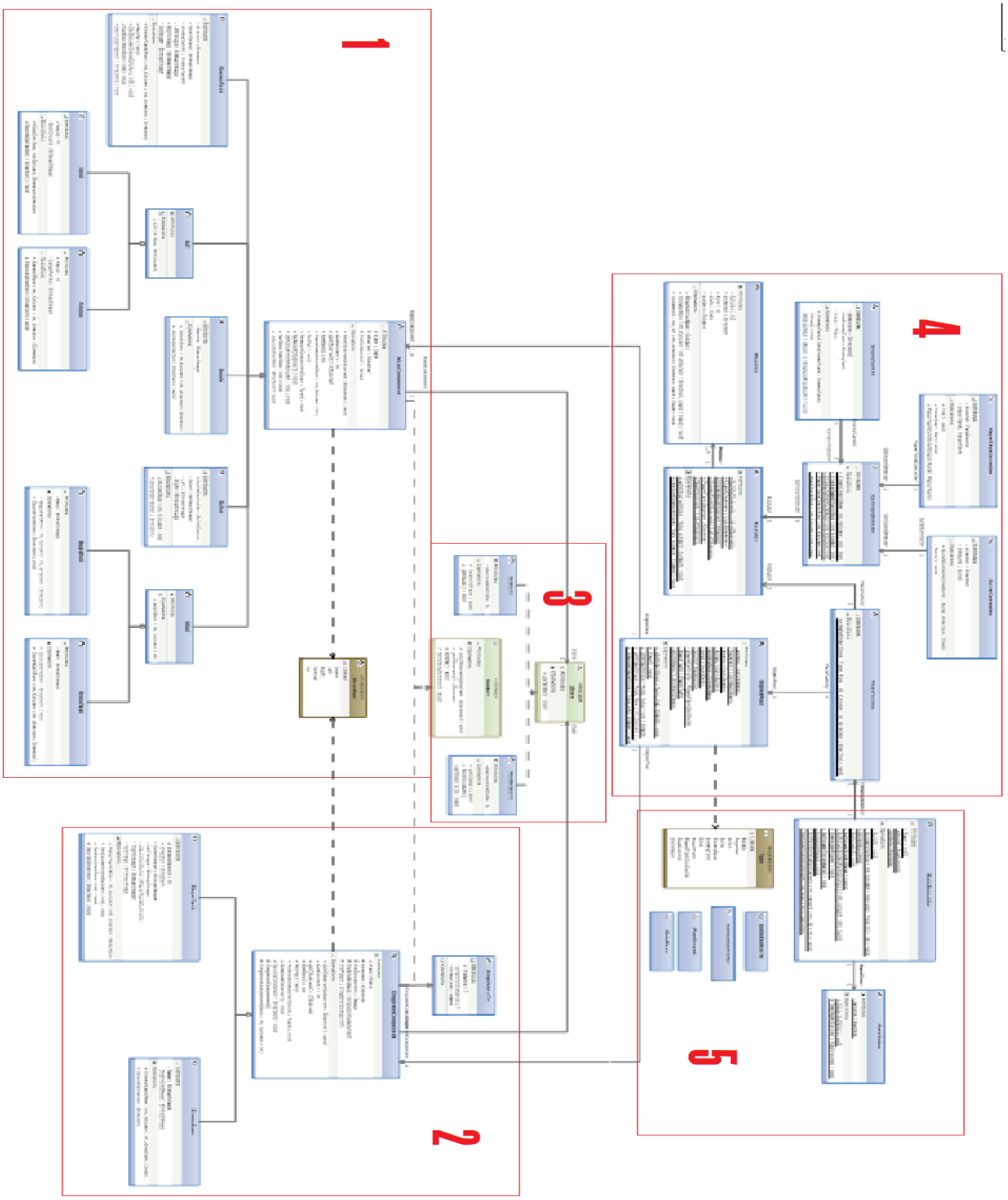
UML Use Case Diagram:



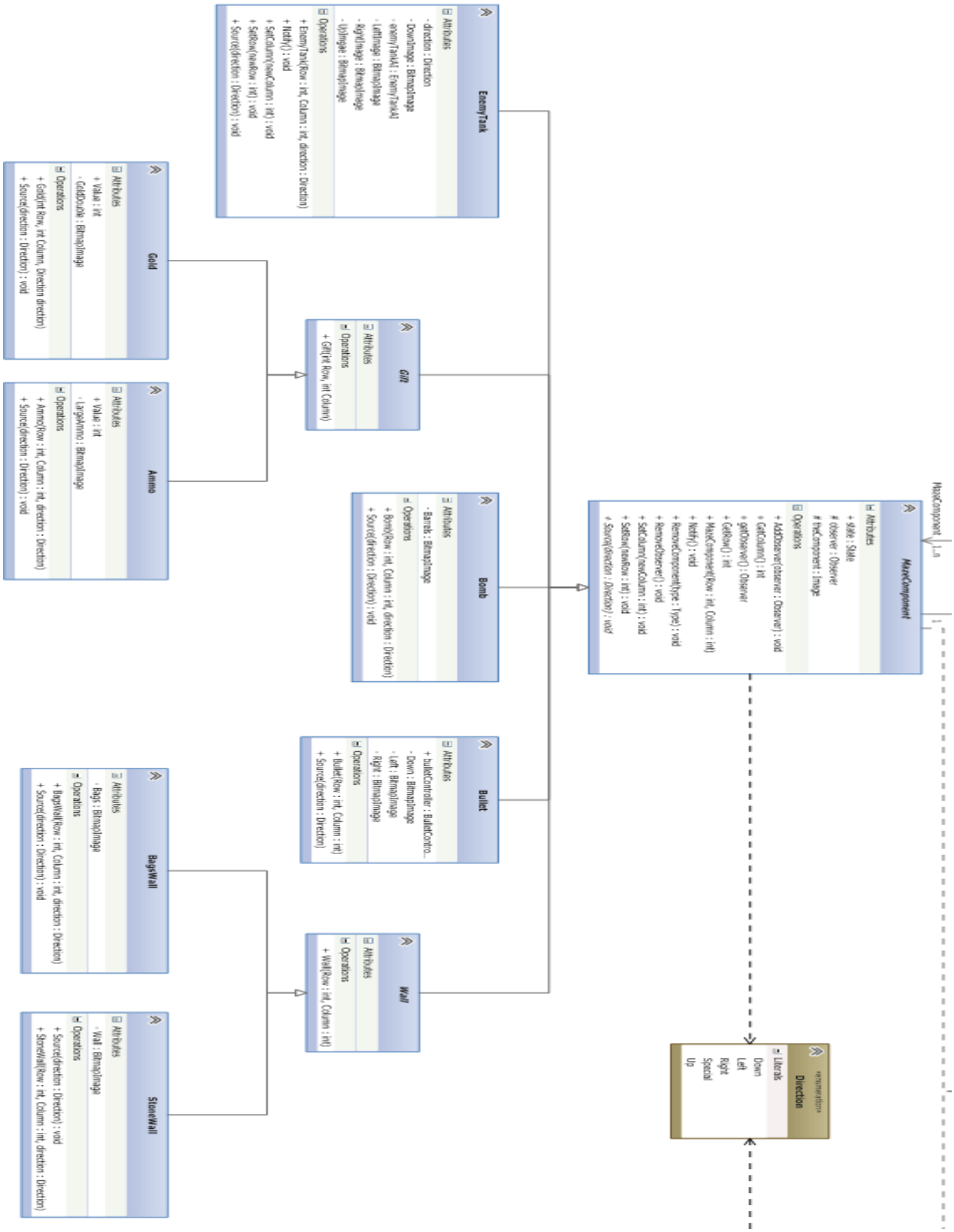
UML Class Diagram:



Due to the large scale of the Class Diagram, we're going to divide it into several parts to make everything clear.



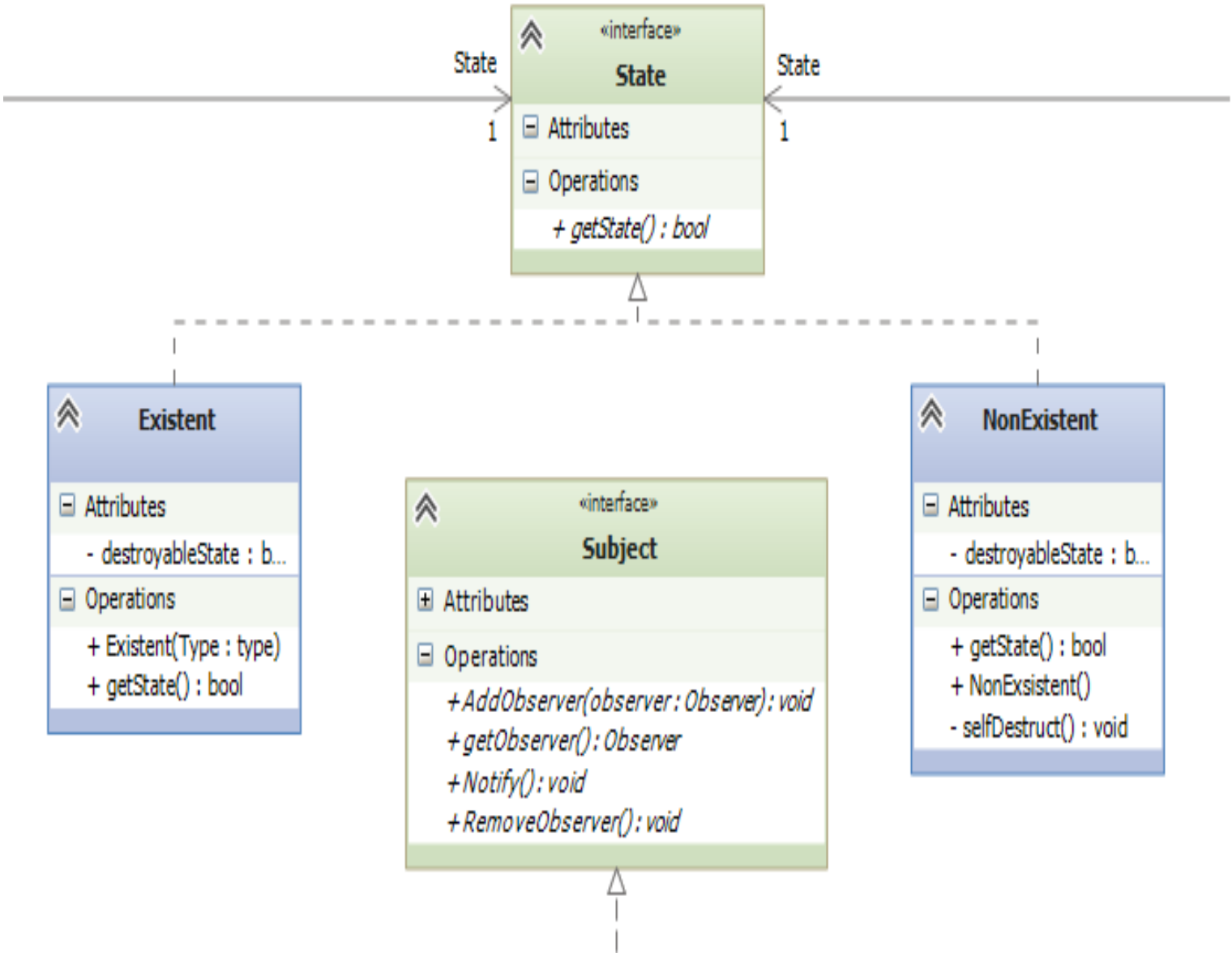
Part 1


MazeComponent
1..n
MazeComponent
Attributes
+ State : State
# observer : Observer
# theComponent : Image
Operations
+ AddObserver(observer : Observer) : void
+ GetColumn() : int
+ getObserver() : Observer
+ GetRow() : int
+ MazeComponent(Row : int, Column : int)
+ Notify() : void
+ RemoveComponent(type : Type) : void
+ RemoveObserver() : void
+ SetColumn(newColumn : int) : void
+ SetRow(newRow : int) : void
+ Source(direction : Direction) : void
«enumeration»
Direction
Literals
Down
Left
Right
Special
Up
EnemyTank
Attributes
- direction : Direction
- DownImage : BitmapImage
- enemyTankAI : EnemyTankAI
- LeftImage : BitmapImage
- RightImage : BitmapImage
- UpImage : BitmapImage
Operations
+ EnemyTank(Row : int, Column : int, direction : Direction)
+ Notify() : void
+ SetColumn(newColumn : int) : void
+ SetRow(newRow : int) : void
+ Source(direction : Direction) : void
Gold
Attributes
+ Value : int
- GoldDouble : BitmapImage
Operations
+ Gold(int Row, int Column, Direction direction)
+ Source(direction : Direction) : void
Gift
Attributes
Operations
+ Gift(int Row, int Column)
Ammo
Attributes
+ Value : int
- LargeAmmo : BitmapImage
Operations
+ Ammo(Row : int, Column : int, direction : Direction)
+ Source(direction : Direction) : void
Bomb
Attributes
- Barrels : BitmapImage
Operations
+ Bomb(Row : int, Column : int, direction : Direction)
+ Source(direction : Direction) : void
Bullet
Attributes
+ bulletController : BulletContro...
- Down : BitmapImage
- Left : BitmapImage
- Right : BitmapImage
Operations
+ Bullet(Row : int, Column : int)
+ Source(direction : Direction)
BagsWall
Attributes
- Bags : BitmapImage
Operations
+ BagsWall(Row : int, Column : int, direction : Direction)
+ Source(direction : Direction) : void
Wall
Attributes
Operations
+ Wall(Row : int, Column : int)
StoneWall
Attributes
- Wall : BitmapImage
Operations
+ Source(direction : Direction) : void
+ StoneWall(Row : int, Column : int, direction : Direction)

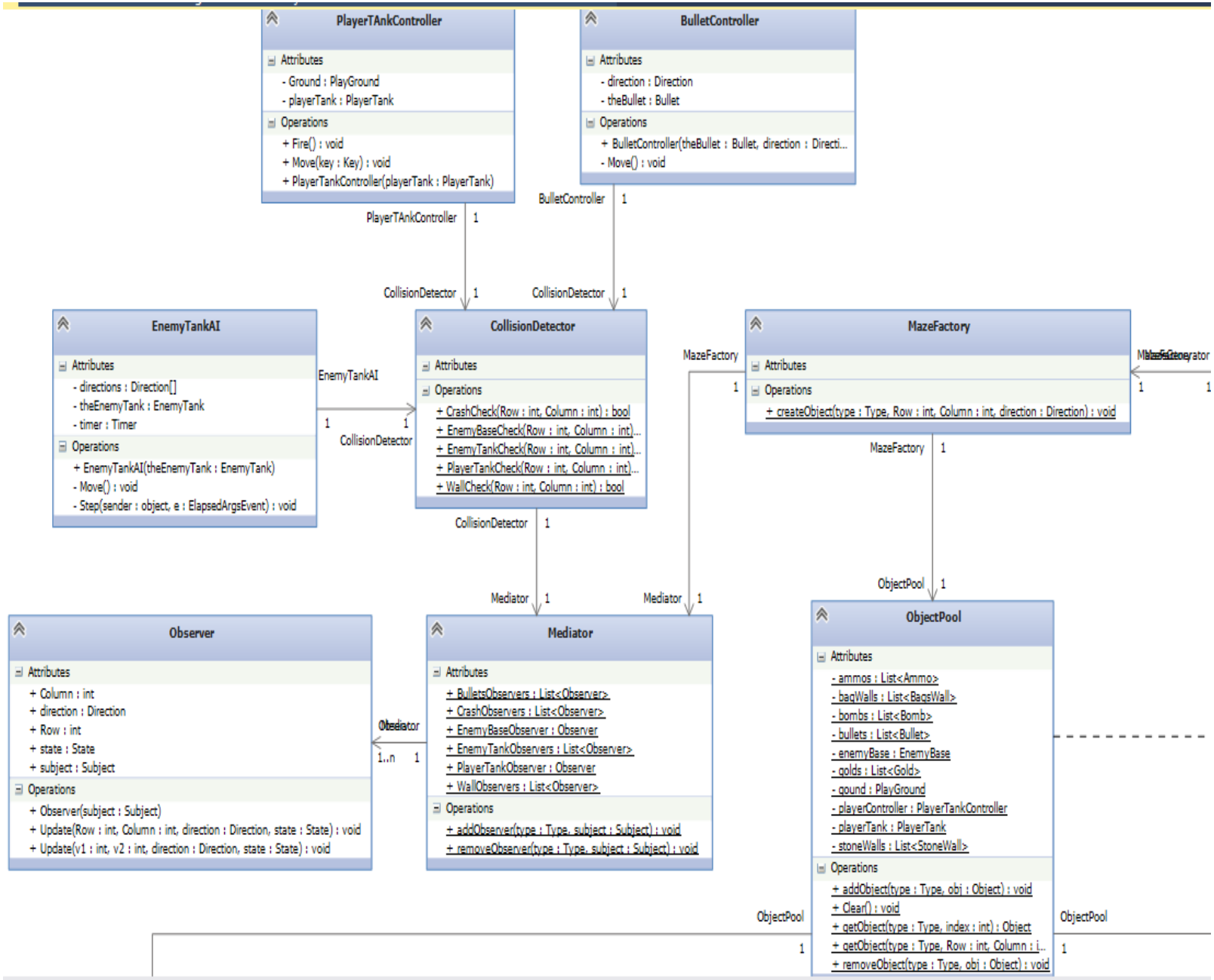
Part 2


Singleton<T>
Attributes
+ Instance : T
- singletonInstance : T
- threadLock : object
Operations
1
SingletonComponent
Attributes
+ state : State
# observer : Observer
# theComponent : Image
# theEnemyBase : SingletonComponent
# thePlayer : SingletonComponent
Operations
+ AddObserver(observer : Observer) : void
+ GetColumn() : int
+ getObserver() : Observer
+ GetRow() : int
+ Notify() : void
+ RemoveComponent(type : Type) : void
+ RemoveObserver() : void
+ Source(direction : Direction) : void
# SingletonComponent()
# SingletonComponent(Row : int, Column : int)
PlayerTank
Attributes
+ AmmoAmount : int
+ direction : Direction
- DownImage : BitmapImage
- LeftImage : BitmapImage
- playerController : PlayerTankController
- RightImage : BitmapImage
- UpImage : BitmapImage
Operations
+ PlayerTank(Row : int, Column : int, direction : Direction)
+ SetColumn(newColumn : int) : void
+ SetRow(newRow : int) : void
+ Source(direction : Direction) : void
EnemyBase
Attributes
- Base : BitmapImage
- DestroyedBase : BitmapImage
Operations
+ EnemyBase(Row : int, Column : int, direction : Directi...
+ Source(direction : Direction)

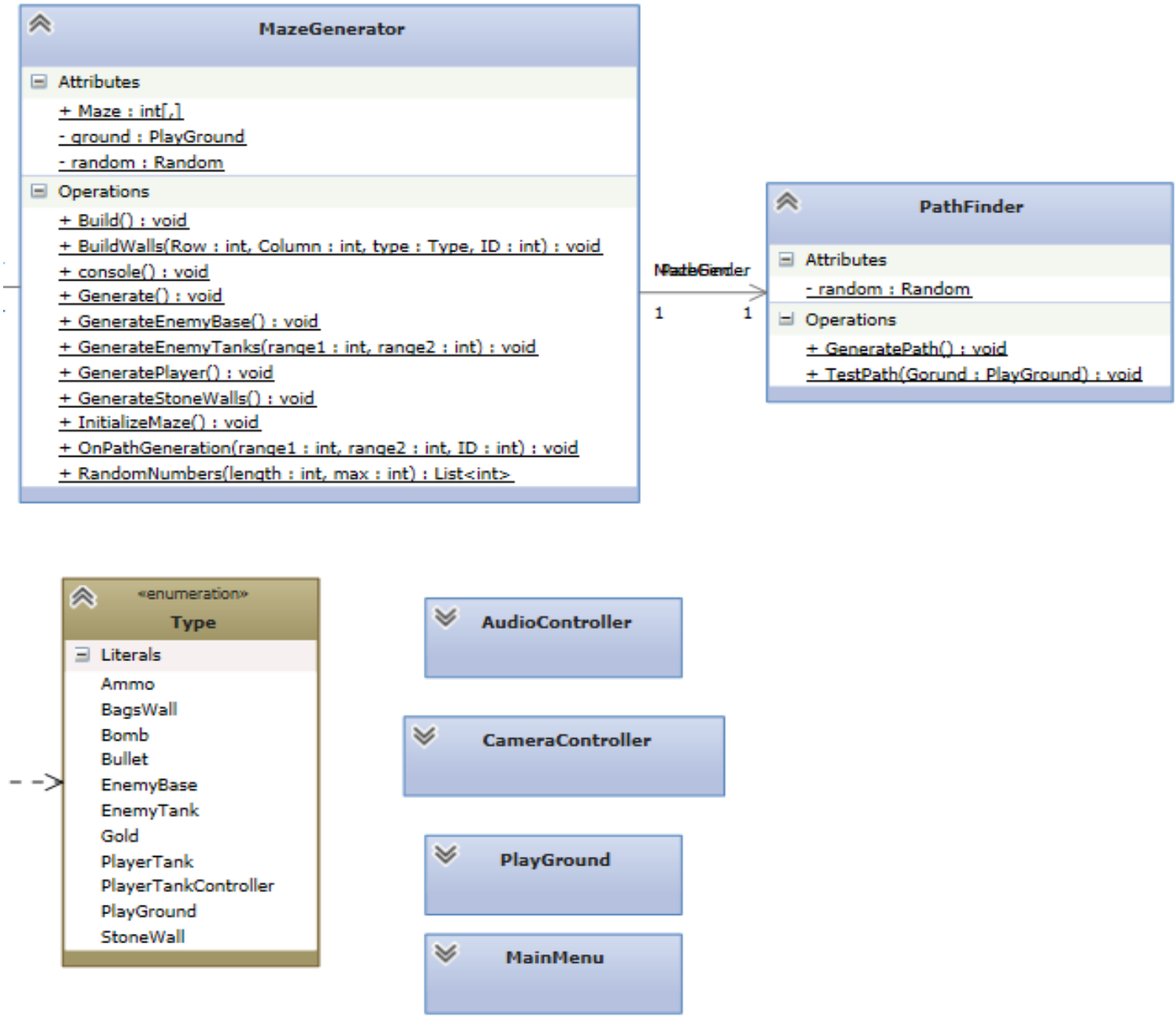
Part 3


«interface»
State
Attributes
Operations
+ getState() : bool
State
1
State
1
Existent
Attributes
- destroyableState : b...
Operations
+ Existent(Type : type)
+ getState() : bool
«interface»
Subject
Attributes
Operations
+AddObserver(observer : Observer) : void
+getObserver() : Observer
+Notify() : void
+RemoveObserver() : void
NonExistent
Attributes
- destroyableState : b...
Operations
+ getState() : bool
+ NonExsistent()
- selfDestruct() : void

Part 4


PlayerTAnkController
Attributes
- Ground : PlayGround
- playerTank : PlayerTank
Operations
+ Fire() : void
+ Move(key : Key) : void
+ PlayerTankController(playerTank : PlayerTank)
BulletController
Attributes
- direction : Direction
- theBullet : Bullet
Operations
+ BulletController(theBullet : Bullet, direction : Directi...
- Move() : void
PlayerTAnkController 1
BulletController 1
CollisionDetector 1
CollisionDetector 1
EnemyTankAI
Attributes
- directions : Direction[]
- theEnemyTank : EnemyTank
- timer : Timer
Operations
+ EnemyTankAI(theEnemyTank : EnemyTank)
- Move() : void
- Step(sender : object, e : ElapsedArgsEvent) : void
EnemyTankAI
1
1
CollisionDetector
CollisionDetector
Attributes
Operations
+ CrashCheck(Row : int, Column : int) : bool
+ EnemyBaseCheck(Row : int, Column : int)...
+ EnemyTankCheck(Row : int, Column : int)...
+ PlayerTankCheck(Row : int, Column : int)...
+ WallCheck(Row : int, Column : int) : bool
MazeFactory
1
MazeFactory
Attributes
Operations
+ createObject(type : Type, Row : int, Column : int, direction : Direction) : void
1
1
MazeFactory 1
CollisionDetector 1
Mediator 1
Mediator 1
ObjectPool 1
Observer
Attributes
+ Column : int
+ direction : Direction
+ Row : int
+ state : State
+ subject : Subject
Operations
+ Observer(subject : Subject)
+ Update(Row : int, Column : int, direction : Direction, state : State) : void
+ Update(v1 : int, v2 : int, direction : Direction, state : State) : void
1..n
1
Mediator
Attributes
+ BulletsObservers : List<Observer>
+ CrashObservers : List<Observer>
+ EnemyBaseObserver : Observer
+ EnemyTankObservers : List<Observer>
+ PlayerTankObserver : Observer
+ WallObservers : List<Observer>
Operations
+ addObserver(type : Type, subject : Subject) : void
+ removeObserver(type : Type, subject : Subject) : void
ObjectPool
Attributes
- ammos : List<Ammo>
- bagWalls : List<BagsWall>
- bombs : List<Bomb>
- bullets : List<Bullet>
- enemyBase : EnemyBase
- golds : List<Gold>
- ground : PlayGround
- playerController : PlayerTankController
- playerTank : PlayerTank
- stoneWalls : List<StoneWall>
Operations
+ addObject(type : Type, obi : Object) : void
+ Clear() : void
+ getObject(type : Type, index : int) : Object
+ getObject(type : Type, Row : int, Column : i...
+ removeObject(type : Type, obj : Object) : void
ObjectPool
1
ObjectPool
1

Part 5



And that's our Class Diagram as Clear as Possible.

Design Patterns:

1. Singleton Design Pattern:

   The Singleton DP is used in the creation of the player tank, and the enemy base as there'll be only one object created from each per run time.

2. Model View Controller Design Pattern:

   The code is divided into three parts: the model where all the data is placed, the view where the playground is created and the controllers where all the controllers of the game are found.

3. State Design Pattern:

   The State DP is used to make existent and nonexistent states for each one of the models. When the model is created, its state is existent and if it's destroyed the state will be nonexistent.

4. Object Pool Design Pattern:

   The object pool DP is the place where all the objects are saved when created and removed when destroyed. All the models are saved in lists of their own type unless it's a singleton component. Once a model is destroyed it's removed from the object pool.

5. Observer Design Pattern:

   Once a model is created an observer is added to it to update any changes and notify all of the other models that a change has occurred whether they are concerned or not. The observer helps in the detection of collision as it tells if two models have collided or not,

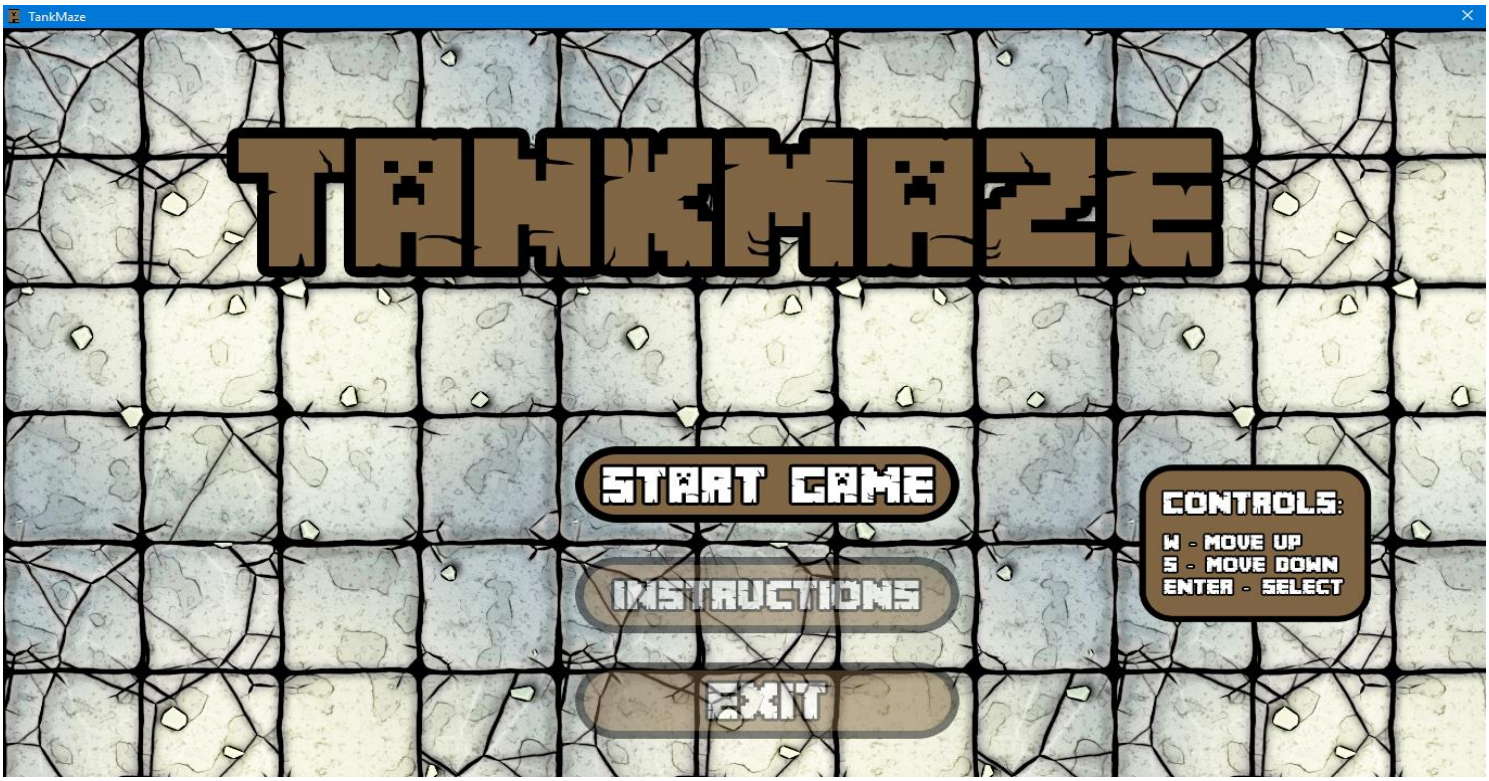
6. Factory Design Pattern

   A factory DP is made to check the type of the object we want to create, add it to the object pool and add an observer to each

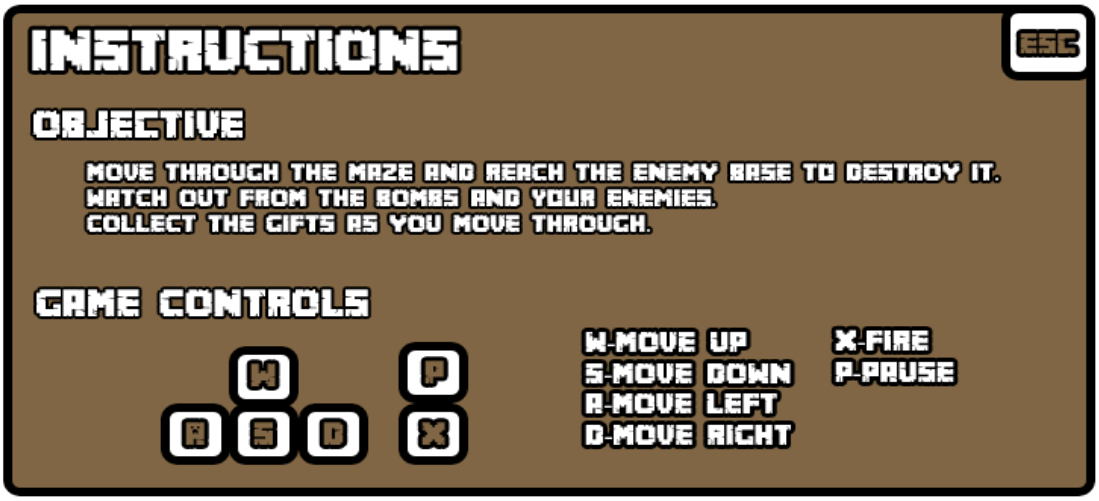
Game Design

The game is designed in a simple and userfriendly way. It consists of:

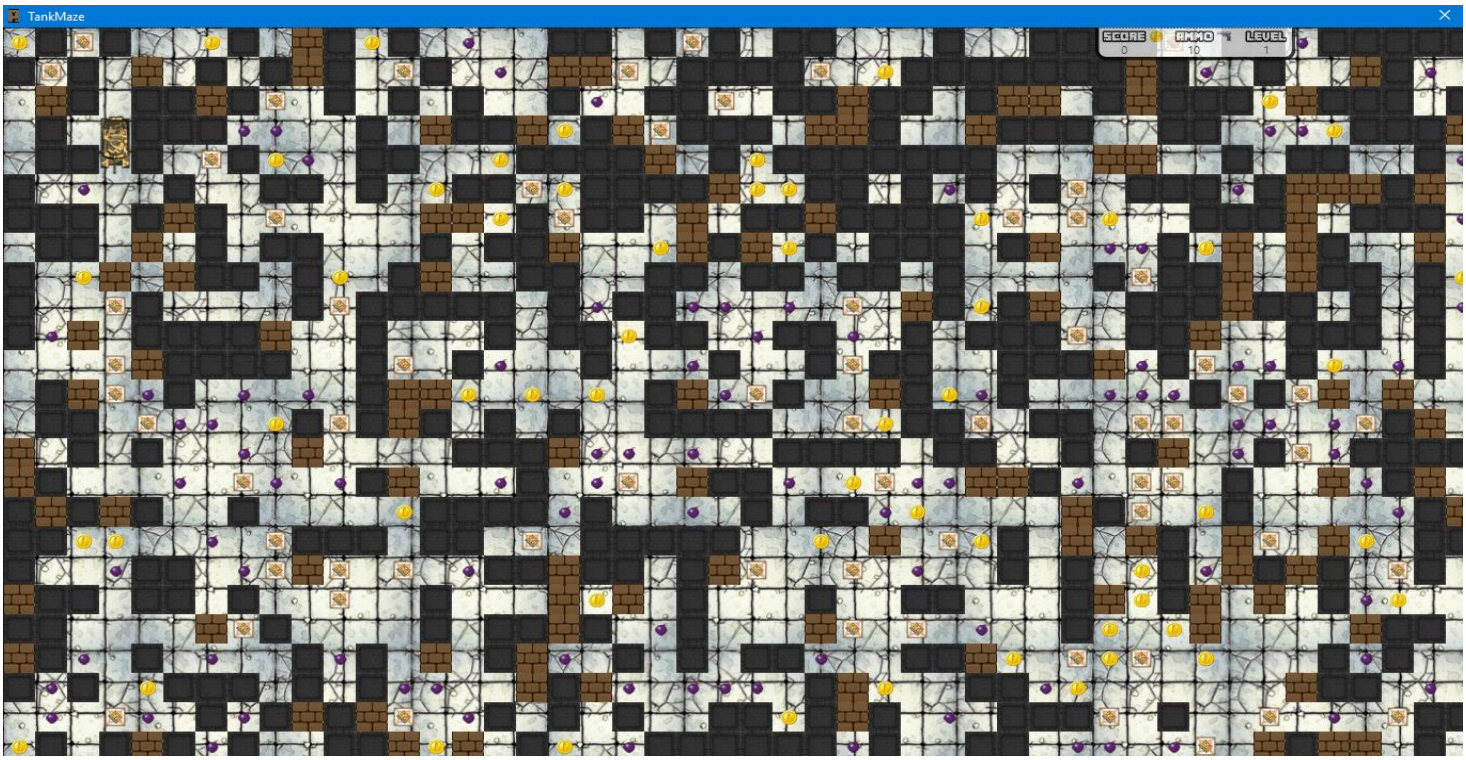
1. Main Menu.

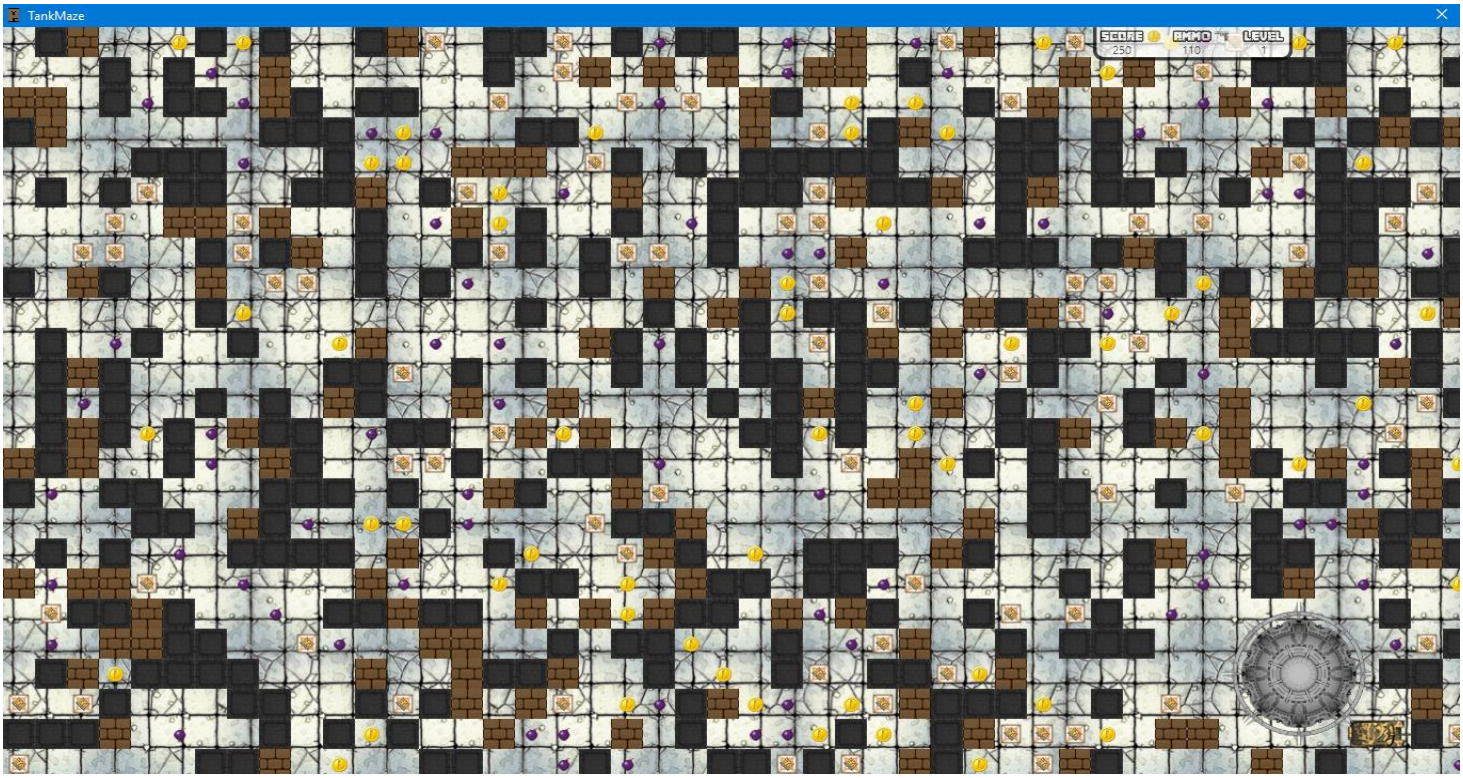


2. Instruction Panel

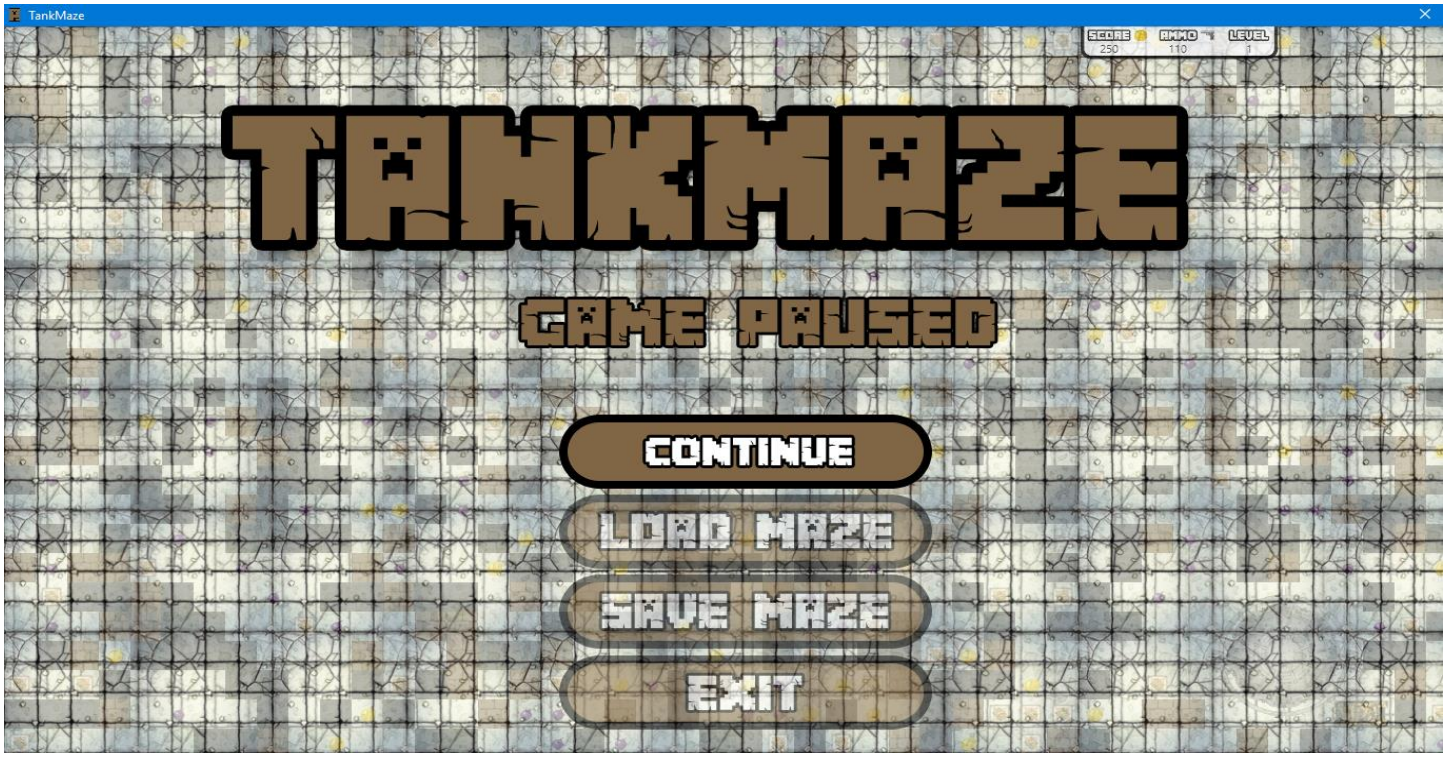


3. The Game

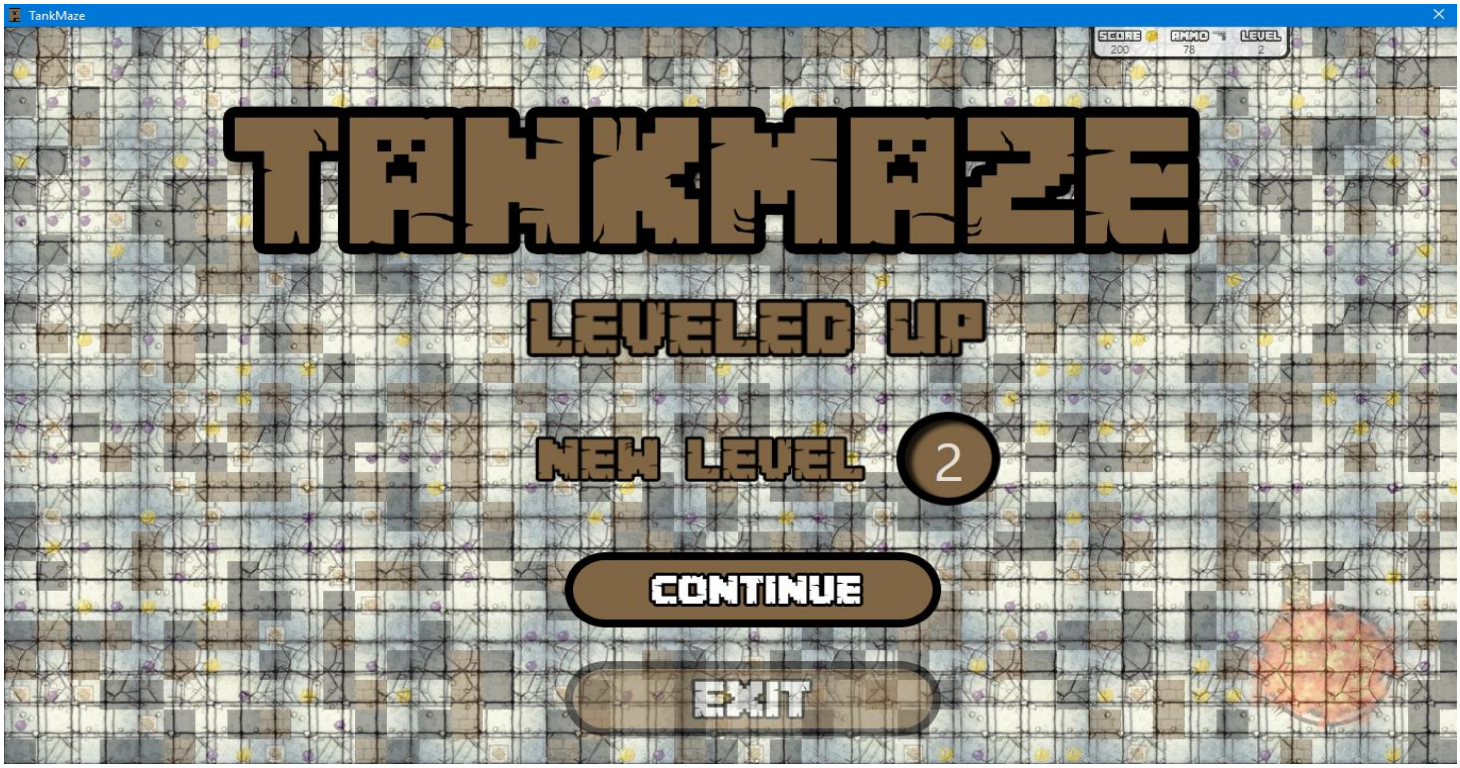

TankMaze
SCORE
AMMO
LEVEL
0
10
1


TankMaze
SCORE
AMMO
LEVEL
250
110
1

4. Pause Menu



5. New Level Menu



User Guide:

Objective: Reach the enemy base safely. Avoid bombs and enemy tanks or shoot them. Collect coins to increase your score. Collect ammo as yours are limited. Shoot the enemy base to go to the next level.

Main Menu:

To select a button, move up or down and press Enter.

To move up, press W.

To move down, press S.

‘Start Game’ starts a new game.

‘Instructions’ displays the instruction panel.

‘Exit’ exits the game.

The Game:

When you start a game the player tank will be placed in the top left corner of the maze, you should move across the maze and reach the enemy base in the bottom right corner of the maze and shoot it. Collect as much gold and ammo as you can to increase your score and ammo. Beware of the enemy tanks and the bombs.

Controls:

W: Move Up

S: Move Down

A: Move Left

D: Move Right

X: Fire a Bullet

P: Pause

Pause Menu:

To select a button, move up or down and press Enter.

To move up, press W.

To move down, press S.

‘Continue’ resumes the game.

‘Load Maze’ loads a previously saved maze.

‘Save Maze’ saves the current maze.

‘Exit’ exits the game.

New Level Menu:

After shooting the enemy base and setting it on fire, you can go to the next level and start playing again.

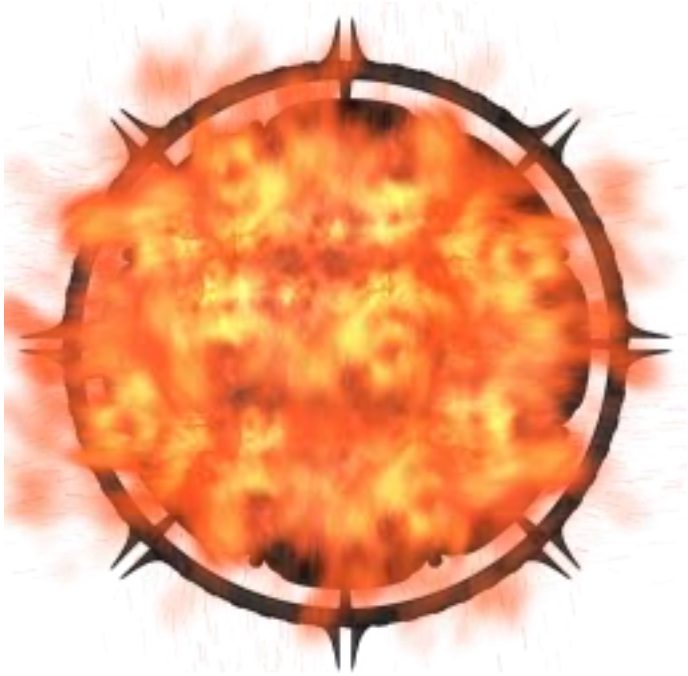
To select a button, move up or down and press Enter.

To move up, press W.

To move down, press S.

‘Continue’ starts a new level.

‘Exit’ exits the game.



THE END