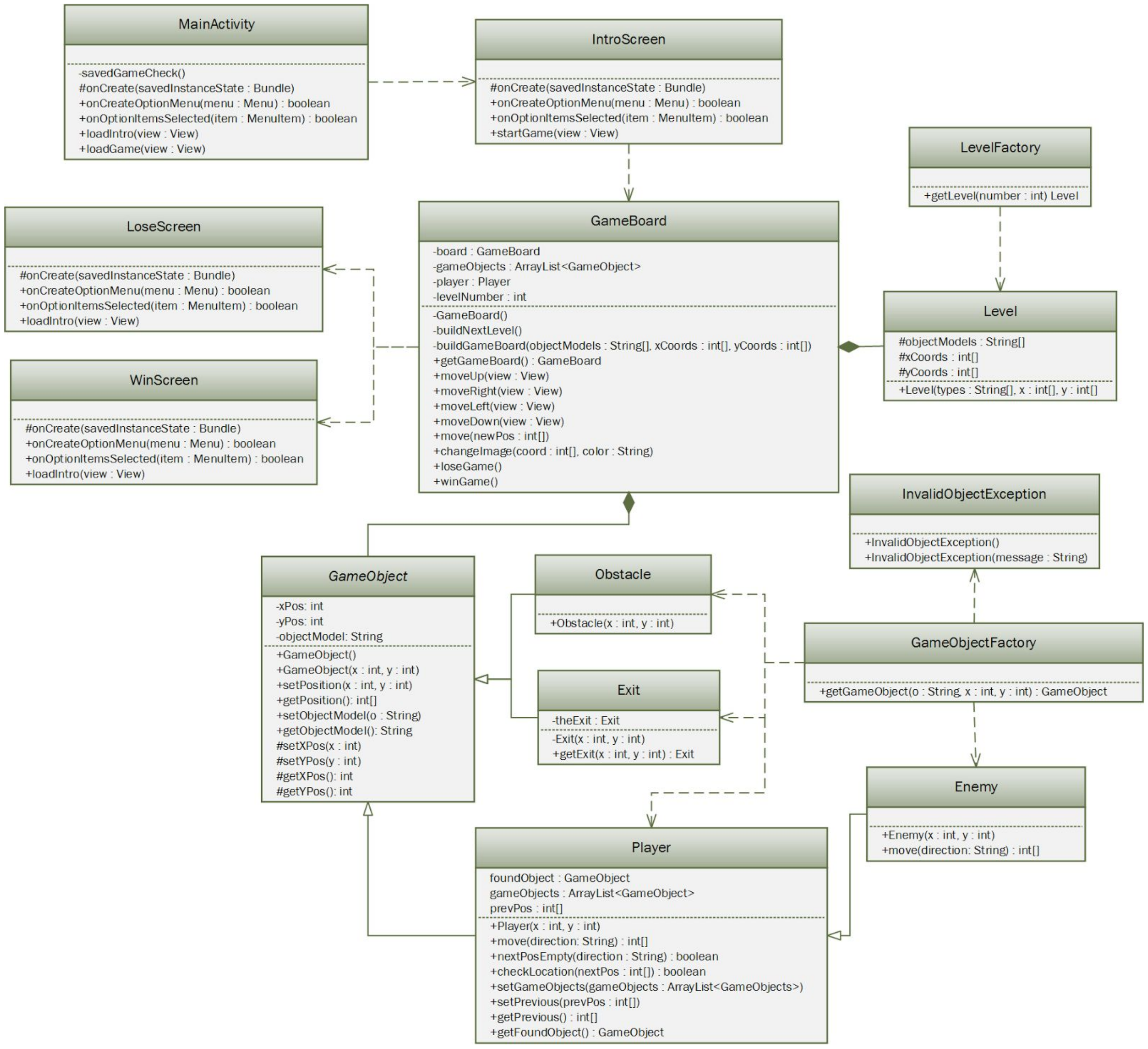
Team: Adam Holt, Joseph Rener, Brent Pivnik, Connor Guerrieri

Title: Escape!

Project Summary: Escape is a puzzle game based on the Android operating system, utilizing a touch interface in order to move a character on the screen. The user's goal is to navigate the character through a maze filled with enemies in order to reach the next level.

1. **What features were implemented and a class diagram showing the final set of classes and relationships of the system. (This may have changed from what you originally anticipated from earlier submissions). Discuss what changed in your class diagram and why it changed, or how it helped doing the diagrams first before coding if you did not need to change much.**

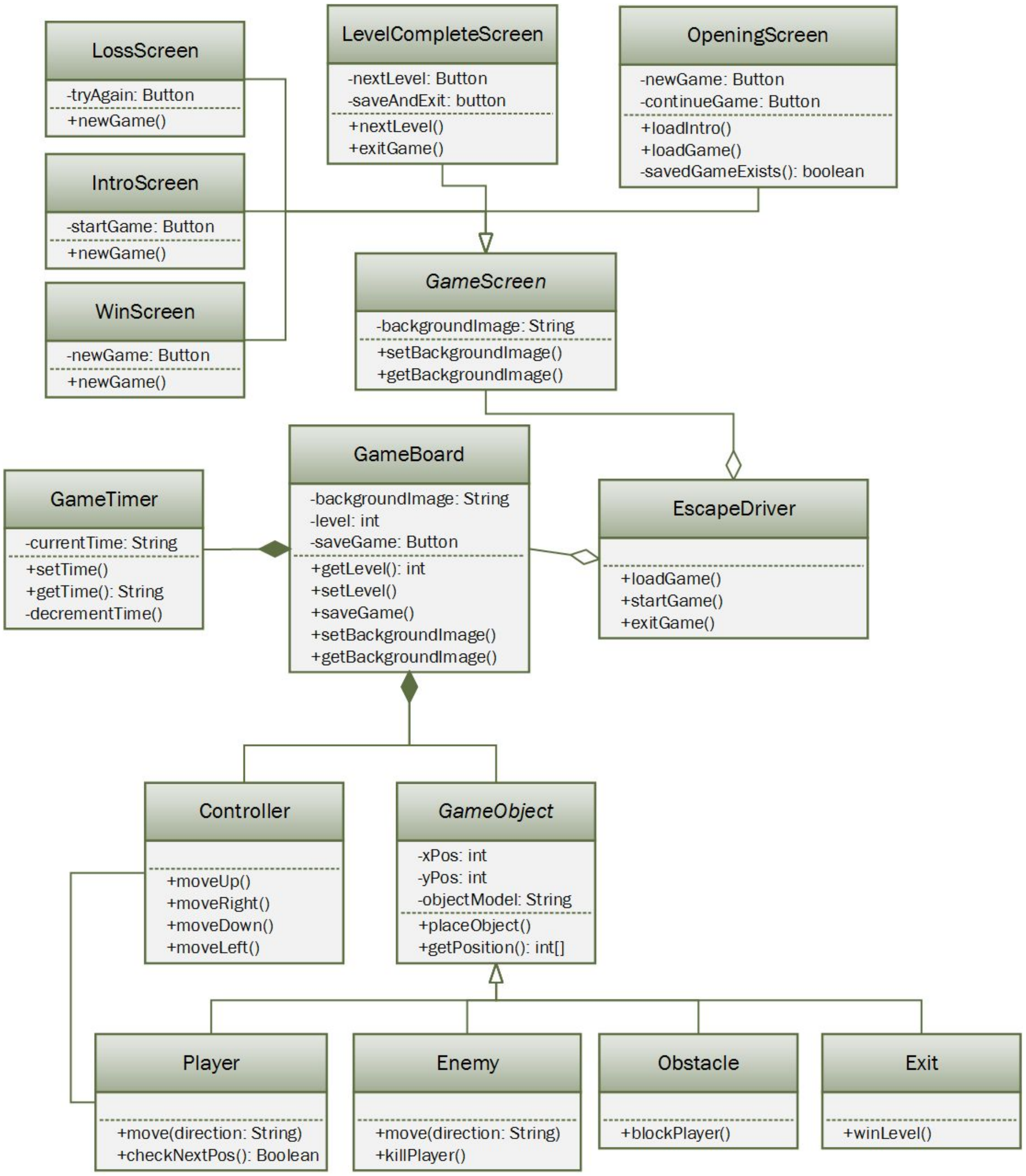

MainActivity
-savedGameCheck()
#onCreate(savedInstanceState : Bundle)
+onCreateOptionMenu(menu : Menu) : boolean
+onOptionItemsSelected(item : MenuItem) : boolean
+loadIntro(view : View)
+loadGame(view : View)
IntroScreen
#onCreate(savedInstanceState : Bundle)
+onCreateOptionMenu(menu : Menu) : boolean
+onOptionItemsSelected(item : MenuItem) : boolean
+startGame(view : View)
LevelFactory
+getLevel(number : int) Level
LoseScreen
#onCreate(savedInstanceState : Bundle)
+onCreateOptionMenu(menu : Menu) : boolean
+onOptionItemsSelected(item : MenuItem) : boolean
+loadIntro(view : View)
GameBoard
-board : GameBoard
-gameObjects : ArrayList<GameObject>
-player : Player
-levelNumber : int
-GameBoard()
-buildNextLevel()
-buildGameBoard(objectModels : String[], xCoords : int[], yCoords : int[])
+getGameBoard() : GameBoard
+moveUp(view : View)
+moveRight(view : View)
+moveLeft(view : View)
+moveDown(view : View)
+move(newPos : int[])
+changeImage(coord : int[], color : String)
+loseGame()
+winGame()
Level
#objectModels : String[]
#xCoords : int[]
#yCoords : int[]
+Level(types : String[], x : int[], y : int[]
WinScreen
#onCreate(savedInstanceState : Bundle)
+onCreateOptionMenu(menu : Menu) : boolean
+onOptionItemsSelected(item : MenuItem) : boolean
+loadIntro(view : View)
InvalidObjectException
+InvalidObjectException()
+InvalidObjectException(message : String)
GameObject
-xPos: int
-yPos: int
-objectModel: String
+GameObject()
+GameObject(x : int, y : int)
+setPosition(x : int, y : int)
+getPosition(): int[]
+setObjectModel(o : String)
+getObjectModel(): String
#setXPos(x : int)
#setYPos(y : int)
#getXPos(): int
#getYPos(): int
Obstacle
+Obstacle(x : int, y : int)
Exit
-theExit : Exit
-Exit(x : int, y : int)
+getExit(x : int, y : int) : Exit
GameObjectFactory
+getGameObject(o : String, x : int, y : int) : GameObject
Enemy
+Enemy(x : int, y : int)
+move(direction: String) : int[]
Player
foundObject : GameObject
gameObjects : ArrayList<GameObject>
prevPos : int[]
+Player(x : int, y : int)
+move(direction: String) : int[]
+nextPosEmpty(direction : String) : boolean
+checkLocation(nextPos : int[]) : boolean
+setGameObjects(gameObjects : ArrayList<GameObjects>)
+setPrevious(prevPos : int[])
+getPrevious() : int[]
+getFoundObject() : GameObject

- We were able to create the basic game as we had originally planned, but ultimately scaled back some of the functionality.
- Instead of enemies moving around the board when the player moves, they have been replaced with stationary traps. The Enemy class still retains the ability to move so that we can implement it later after solving the issues with enemy movement that we experienced.
- Our original hope of having procedurally generated levels was scaled back due to time constraints, and instead we created four static levels.
- Various methods were moved between classes where it made more logical sense for one type of object to handle the game logic than another. This includes consolidating the Controller object into the GameBoard class.
- Additional methods were added in order to simplify what each method did or to extend the functionality of a class.
- Some methods and classes were added in order for the original diagram to work with the Android framework.
- The LevelFactory and GameObjectFactory were added as we realized that it would be more efficient to generate these object through a factory method.
- We removed the Timer class from the final app, but left the ability to add it in a later iteration.
- Even though the changes to the class diagram were non-trivial, it was still very helpful to have the original in the process of coding the game. As we implemented the original code, we were able discover new and better ways to handle the game logic. Having the original structure in place made the initial coding far simpler.

2. **Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?**

- We used the Factory Method design pattern in implementing the GameObjectFactory class. The getGameObject static method returns one of four different objects that inherit from the GameObject class. This allows for polymorphic references to GameObjects to be included in an ArrayList that is used to populate the GameBoard with the necessary GameObjects.
- A slight variation on the Factory Method design pattern was used in creating the LevelFactory class. The getLevel method takes an int as a parameter and returns one of four static levels which determine the layout of GameObjects.
- Both the GameBoard and Exit classes are instantiated as Singletons in order to prevent accidental creation of multiple instances.
- The GameBoard object is a composite of several instances of GameObject and Level.
- In our original plan to have Enemies move when the Player did, we would have implemented an Observer design pattern so that user-selected moves were broadcast to any GameObject with the logic to move.

3. **In addition, the report must discuss how the final system changed from the design you presented in Project Part 2. In particular, include the class diagram you submitted for Project Part 2 and use it to compare and contrast with the class diagram that represents the final state of the system.**



- Added GameObjectFactory and LevelFactory
- Delegated knowledge of level construction (GameObject placement) to a Level class
- Added MainActivity Android class as well as necessary Android methods in GUI classes - replaced EscapeDriver class

- Unlike the original EscapeDriver which was aggregated of the GameBoard and various screens, the MainActivity class merely triggers the instantiation of the game while the activity itself is carried out by the GameBoard and various Android framework elements
- Consolidated Controller into GameBoard class
- Removed logic for ending (win or lose) game from GameObject classes and assigned the logic to the GameBoard
- Removed the LevelCompleteScreen
- Removed the abstract GameScreen class - GUI game elements are handled by Android
- Removed the GameTimer class due to time constraints in adding it to the GameBoard
- Created and InvalidObjectException class which extends Exception in order to handle requests to the GameObjectFactory for nonexistent GameObjects

4. **What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?**

   We've learned quite a bit after completing this process. Namely, that designing the system entirely before writing a single line of code adds a tremendous amount of overhead to the project. In fact, in our case, it added weeks to the creation process. That being said, it made for a very different experience once we finally got into the programming aspect. We were able to split up work based on class, and even though the system was split among 4 people, we all had a very clear plan as to what we need to do and how all of the pieces fit. On top of that, once the backend code was done, overlaying the frontend was infinitely easier than it would have been without the analysis and design process. These things lead to a somewhat magical moment when the various classes came together and worked. Ultimately, the large amount of overhead was well worth it when implementation came around.