Ian Cullum
Dane Buckley

Our design rationale includes a series of loops that completes when the game concludes. We have a dayloop which controls the day cycle and loops when the day has been completed (9 scenes have been wrapped), a turnloop that determines what player moves and what moves they can make, and finally a gameloop which controls these loops and scores the game at the end. It is all controlled through a GUI made in JavaFX and primarily uses buttons to take user input. The loops use seperate manager classes to accomplish their goals and we have separated the managers (such as movemanager, scoringmanager) in an effort to implement object oriented design and have more organized code. As far as our coupling goes, External coupling is inevitable as we are parsing XML files. In addition to this, stamp coupling occurs whenever we have the “Players” array, areabank, setbank, scenebank or any data structure being passed from class to class within the model. Although many parts of our program are not functionally cohesive, we do have many forms of sequential cohesion and procedural cohesion. In addition to this, we experimented with implementing package private variables to further our level of encapsulation. While on the topic of design patterns, there were many opportunities to use method overloading that we chose not to take however in some cases it was the most efficient method of organization. For functions like notifPrompt and even triggerButton are both methods that will need to be used in different ways based on the information we have at the time. Many forms of inheritance were implemented, such as in our Room class, it is a parent to standard rooms, but also the special rooms such as the Casting Office so we can give them special properties. Moving on, we chose to implement MVC as it was the most appropriate design pattern given the constraints and requirements of the project. It was also a really good way to practice. It used gameloop as the model, the Controller/UI as the controller, and the gui as the view. We also made the decision to only couple UI and Controller with gameloop, so they are completely separate as you can see in our class diagram. We did this to clearly separate the Model from the View, and not allow for any unwanted input to make its way in.