Kevin Gottfried

Labyrinth Final Report

This program is an implementation of the popular board game *Labyrinth* in Java. The program features two different game modes; computer and human. Player 1 has a blue pane and a blue token on the board and the computer or Player 2 has a green pane and a green token on the board. In computer mode, one player faces the computer. When it is the user's turn, the GUI displays the user's pane which includes their player number and color, the extra tile they have to insert and rotation buttons, their current treasure card they need to capture, and their number of completed treasures. The board is surrounded by arrows pointing to the rows and columns where the user is able to insert the extra tile piece. After the user inserts the tile, (the user is prompted to select a tile to navigate to, or they have the option to skip their turn via a button on the bottom of the board.) If the user attempts to insert the tile in a row or column that would counter the last tile inserted or if they try to move to a tile that is not reachable, they are alerted and asked to retry their move. After the user takes their turn, a display shows that the computer is taking their turn. In human mode two players face off head to head and switch back and forth taking turns. When it is a user's turn, the opponent's pane is hidden making it easier to see who's turn it is as well as hiding the other player's treasure card to make it more difficult. The first person to collect all 12 treasure cards in their deck in the order they are displayed is the winner. Once a game is won a pop up window appears asking the user if they want to quit or if they want to start a new game. If a new game is selected everything resets and a new game starts.

In our first team meeting, we introduced ourselves and discussed what implementation of the final project that we would like to pursue. After we decide on Labyrinth and Java as a language, we decided that our method of communication was going to be through Facebook and that we wanted to use GitLab as a source control. As well we created a Google drive to share and store documents and files to be used for the deliverables and design specs for the project. Our second meeting was one of our longer and more important meetings and the biggest step was developing a full understanding of the rules of Labyrinth, so we spent some time talking about how to play the game. In doing so, we were able to sketch out the different components of the game that we thought would be necessary for our implementation. Attached (Figure 1) is a mockup of the GUI that John designed during our second meeting. Having a mockup of the GUI was important because it allowed us to create an understanding of the initial flow of the game allowing us to further delve into specifications and develop a deeper class hierarchy. As far as developing the GUI, we chose to use JavaFX because 2 of the 4 members of our group were experienced and felt that we would have a stronger and more efficient build. John and Alex also had pre-existing code from a previous project using JavaFX that we were able to reuse to give us ground to stand on. During this meeting, we created a UML diagram that sketched out our class hierarchy which allowed us to conceptually piece together the UI and the back end (Figure 3). Ultimately in retrospect, this UML became far more advanced and the underlying architecture of our game loop changed a few times, but we were able to use the initial UML to build a working version very quickly. For deliverable 2, we were required to create specifications, task breakdown, and system design documents, but as well we

decided to take it as an opportunity to extensively detail our project, rather than just attempt to fulfill a requirement. We created a very detailed and fluid design specs sheet that outlined almost every requirement and implementation that we needed to reach for the whole project. The task sheet was ordered by theory of relevance and dependence to each requirement. We also took the time to break down each component and assign respectively, the categories and requirements that would suit each of us best for our sections. We color coded each section and respected author to better organize and distinguish between each task. In a general application, John and Alex worked mostly on the GUI and the event-driven aspect of how the back-end and game flow would interact with the UI, and Eric and I worked on designing the architecture and structure of the back-end. From these two major parent components, we created a GUI, Board, Tile, and Button class that allowed us to manipulate our GUI in the desired manner. We as well created Player, Card, Deck, Path and GameState classes to handle the game logic. GameState consisted of enumerators that we used to detail which state the game was in and who the current player was. The player class was used essentially to display the player pane which held information like the treasure hand, extra tile to be inserted, and number of successful treasures. The path class was utilized to handle our path finding algorithm, which Eric genuinely created and implemented very successfully. It utilized a breadth first search and returned an array of possible paths that could be taken, for the user to choose from. Although we did not necessarily explicitly discuss it, we attempted to implement an iterative development process, and utilize each deliverable to integrate and test the code that we had. Throughout the course of the project, we had weekly status updates, where we met as a group for about 15 minutes

either before or after class, and discussed the progress that we had made individually throughout the last few days. During this time, we also were able to finalize our merges and make sure that everyone's code and projects were up to date on their local machines. We took these opportunities to update our task list and discuss some of the conflicts that we were having both personally and as a group, either due to a mental block or lack of resources, possibly from another team member missing their part. As a group, we had 2 major sit down meetings, where we dedicated a few hours of our night to work together all at the same time to design and code our system. These meetings turned out to be very efficient because we were able to use peer programming to tackle segments very quickly. As well, we could communicate very efficiently and succinctly, allowing us to work through some kinks in our architecture. The first major meeting that we had, we both built the GUI and player classes, and integrated them together. The second major meeting that we had, was one of the last meetings that we had. We spent 3 hours finalizing and finishing all of the requirements that were left on our task list, and were able to piece together and integrate the final components of our system. This meeting was one week before the due date, and allowed us extra time to fix and test any final bugs that we may have found over the following week. This was one of our more efficient meetings, and we managed to create and implement the Computer class, a more advanced GUI, and fix certain bugs regarding treasure card collection.

I found that through some of the meetings that we had, the most efficient way to organize and detail our system and requirements was to verbally try to explain the game flow and logic to the other team members. In doing so, we could follow a logic path through the system and along the way, point out different use cases and possible

design paths to test and take. This was essentially how we created our task and requirements list (figure 4). Since we took an iterative approach, we were able to dynamically and successfully update our task list throughout our development, although we did not need to make many changes down the road. As per our communication method, I personally vouched for Facebook as I had used it in the warmup project and considered it to be a very successful form of communication. The group seemed hesitant at first, but I think by the end of the project they all seemed to agree that it was more efficient than email or text. As far as our meeting locations, we met before or after almost every class in the Perkins lobby except for our final meeting. I suggested that we meet off campus for a change of scenery in New Moon Café, where we spent almost 3.5 hours integrating and testing our final system. It seemed that this too, was an efficient but seemingly simple aspect to team productivity, but a good change of scenery seemed to boost morale and allow for an increase in workload. I think that I would have changed our testing and debugging standards for this project and possibly future ones as well. It was a total contrast compared to the last project I worked on, where I put many hours into debugging and testing. Although I felt as if we put in a sufficient amount of time testing and integrating, I think that we did not necessarily exceed the amount of testing that I would a preferred. I think that we spent most of our time our allocating to different resources, that we did not develop the most sophisticated test plans and this is something that I would like to change for future development. I appreciated the benefits of using an iterative approach as oppose to the waterfall method that I used in the first project. I definitely noticed the benefits in design and build, where we did not necessarily need to make sure our specifications were 100% to

start. I think that during this project, we worked more efficiently and had a final product a lot quicker than we did during the first project, and I think that this is partly due to the nature of the iterative process. I think that with regards to testing, the iterative process allowed us to do more unit testing, which in turn made it more efficient and easier for us to integrate and test our whole system. Overall, I think that we had a very successful implementation and build for our Labyrinth project. I am satisfied with the results that we produced and the process that we used. As a group, I think that we worked very well together, and did not seem to have many issues. We all shared a decent workload and were able to find different aspects of the project for everyone to evenly distribute work. I think that as far as our UI goes, we tried to keep a simple and maintainable design. Designing and implementing the GUI was a lot more difficult and tedious than we had thought, and required a lot of planning and a lot of lines of code. At first, it was difficult to keep our file systems organized and get everyone onto the same page, but once we got over the hump, we had very minimal issues. I think that it was very difficult to maintain and well-documented system, but since we communicated very efficiently and effectively on Facebook, I think it allowed for a lack of comments. In the end, we spent time organizing and commenting our code, as well as performing a cleanup of dead code. As a whole, I think that this project was definitely difficult at times, but as a team we were able to balance each other and successfully complete Labyrinth on time.

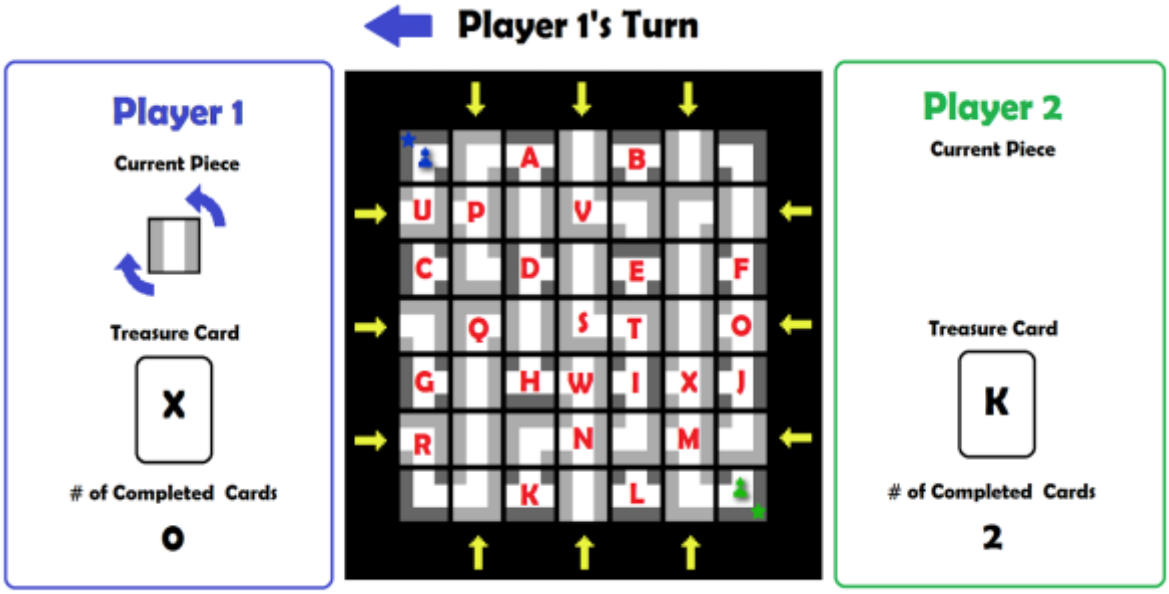

Figure 1

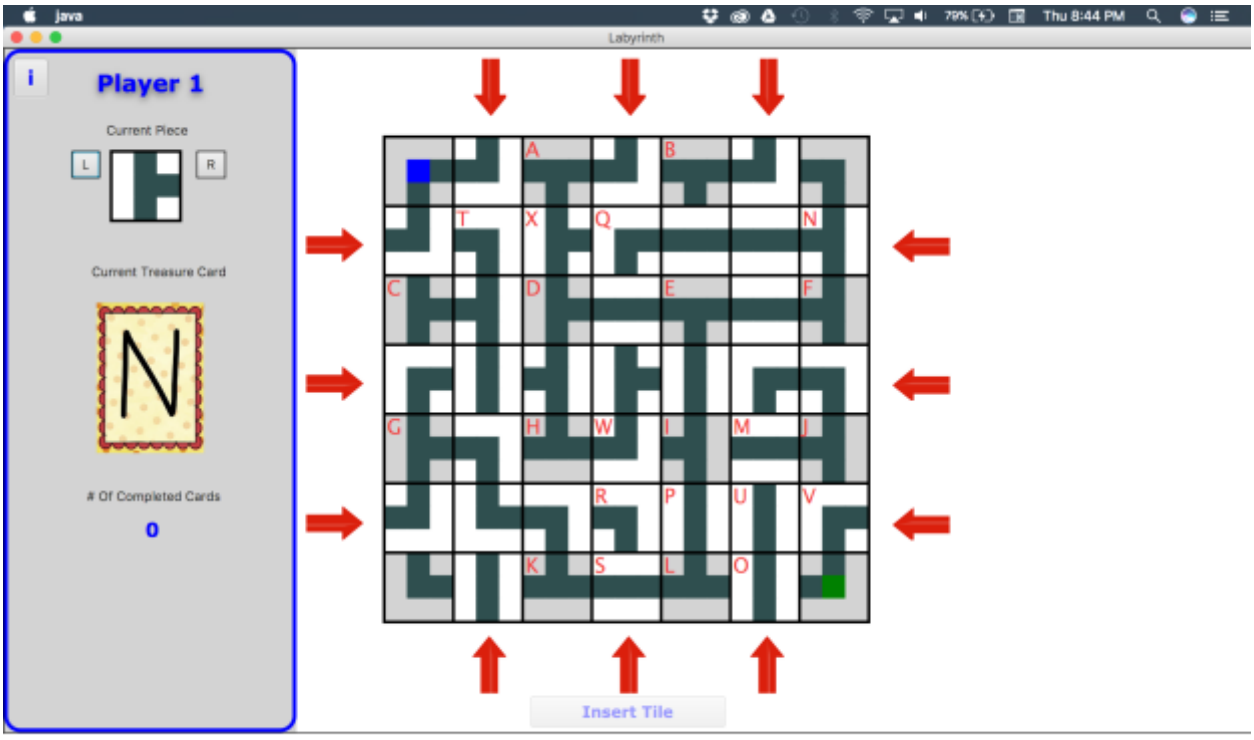

Figure 2

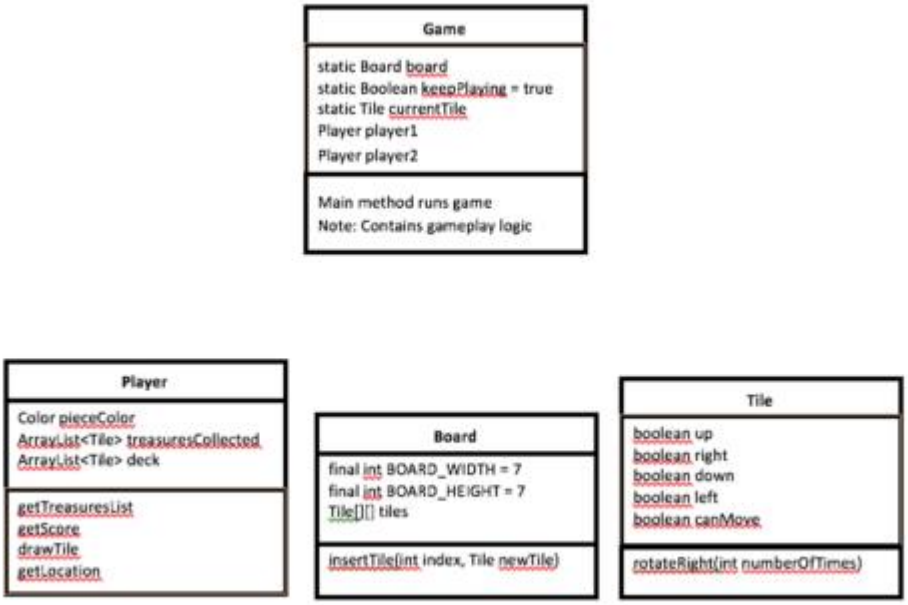

Figure 3

Req ID (Order)	Requirement	Task ID	Tasks	Owner	Importance	Difficulty	Component	Implementation	Test	Integration
A (1)	Game board is a 7x7 grid of square tiles	1	Create game board model representing 49 tiles.	Alex	High	Easy	Model			
		2	Create gameboard graphic.	Jon	High	Easy	View			
B (2)	Each square tile contains a path which connects two or three sides (see specification for tile types and quantities)	3	Create tile model.	Alex	High	Easy	Model			
		4	Create tile graphic.	Jon	High	Medium	View			
C (3)	Two player game	5	Create Player model	Kevin	High	Medium	Model			
		6	Create Human Player type	Kevin	High	Easy	Model			
		7	Create Computer Player type.	Kevin	Medium	Easy	Model			
D (4)	Sixteen tiles are non-movable - see specification image for tile types	8	Initialize non-movable tiles as appropriate tile types and orientations.	Eric	High	Easy/Medium	Model			
		9	Prevent certain tiles from being moved	Eric	High	Easy	Model			
		10	Represent non-movable tiles differently	Jon	Low	Easy	View			
E (5)	34 tiles are movable - 16 corner, 6 tee, 12 line	11	Initialize movable tiles as appropriate tile types	Eric	High	Easy	Model			
		12	Place 33 movable tiles on the game board randomly with random orientations at the start of the game	Eric	High	Easy	Model			
F (6)	One movable tile is not on the game board (extra maze tile)	13	Designate an extra maze tile	Alex	High	Easy	Model			

Figure 4