

Dungeon and Pacman

Peter Joe | Dennis Cheung | COMPSYS302

# Minimum Requirements Checklist

1. When the game is opened, there is a welcome screen with options to start and exit. Once start is selected with either keyboard or mouse, they are brought to a mode select, where the player can play single player or local multiplayer.
   1. For single player, there are 3 levels, each with 4 enemy AI which each have individual “personalities”.
   2. For multiplayer, there is 1 level, which the user(s) can select their character model before the game starts.
2. When the game starts, the player and AI are frozen in place until the countdown timer counts down to 0. An ImageView is created and put in a Pane every second in a Timeline we created, this shows the countdown.
3. A “START!” message will appear signaling the player and AI can move. There is a Boolean for running, and if the game is “running” then the game will move. The game initially starts as false and becomes true when the game “START!” message plays.
4. The window is not resizable, and is locked at 1080 x 800. We set the stage in main.java not be resizable.
5. When the user collides with our coins, then the player score increments. Initially we created an arrayList for all the coins, if the user intersects with a coin, we set that specific coin’s “alive” property to false, and then we check through all the coins to see it’s “alive” property is false, if it is then we remove it from the array and the game to make it disappear.
6. When the user collides with a wall, they will not be able to pass through it, the player is also able to wrap around the window as it’s X position is set to the other side of the map.
7. When the user collides with an enemy, the user will the spin around until they vanish and their number of lives decreases and it shows on the user’s life bar decreases on the top left. The user spinning is a gif and we have a Timeline that decreases the opacity of the user, and the hearts in the hearts arrayList will decrease.
8. We have a countdown from 3 minutes, we deviated from the requirements because the level and AI were too complicated and it took too long to beat a single level within two minutes. The timer was made the same as the countdown earlier with a Timeline. When KeyCode == PgDn, then the time is set to 0.
9. When KeyCode == P, then running becomes false, so the game doesn’t update thus pausing the game, if pressed again then running becomes true so the game resumes. When KeyCode == ESC, then a confirmation screen appears to confirm whether the user wants to exit.
10. When KeyCode == PgUp, it automatically declares the floor to be cleared and displays a victory screen.
11. When the player collects all the coins, then running becomes false, stopping the game, and we load an FXML file for the victory screen.
12. We created a function for the SFX using a JavaFX library AudioClip. And we use the function whenever it should happen.

## Improved functionalities

1. Story
   1. We created a story to accompany the game, the script is in the form of a string array.
2. Narrative (Visual Novel)
   1. We have images we put over a background, then we read the script array, the text is in the form of “#Text”, the # identifies the character and the text is the dialogue. We have two methods, both with substring the string in the arrays.
3. Attributes
   1. The character class contains the attributes. The constructor sets the default values for each of these attributes, the rest of the class contains setter, getters, incrementors and de-incrementors of different attributes.
   2. On the upgrade screen, we have buttons which trigger the incrementors/de-incrementors.
   3. These attributes affect the gameplay, HP increases the amount of lives the user has, the SP increases the speed of the user, the Power Duration increases the time your character can use the sword.
4. Boss Battle
   1. The boss is a bigger version of the typical enemy, it will be killed when 12 normal sized enemies are killed.
   2. The health of the boss is shown with an HP bar which decreases as the number of enemies are killed.
   3. The size of the boss is refreshed whenever an enemy is killed.
5. Multiple Levels
   1. Our levels are created with an int map. When we initialize the game, we go through the entire int map, and find what the values are in the map.
      1. ‘0’ = Empty space, ‘1’ = Wall, ‘2’ = Coins, ‘3’ = Swords.
   2. When the win condition or the game over condition is reached, then we have a levelSelect class which increases the player’s current level and switches the FXML to the Victory Screen and the then to the next story and level gameplay or displays a game over screen with a retry button depending on the result of the level.
6. Character Creation
   1. When new campaign is created the user has an option of choosing a character, when the character model is clicked on or enter is pressed, the model in the player class is set to that specific character and it is carried onto the gamplay.
7. Functioning AI
   1. As mentioned above, there are four AIs in each level with its own “personalities”. Personalities determine target tiles for AI in order to perform “trapping”. However, constant pursue would tire out player will to continue and lose interest in the game. Therefore, we have implemented two different modes “scatter mode” and “attack mode” for AI to allow player to take a break.

During scatter mode, each AI will return to their respected corners and circle around it until attack mode is on, which then their personalities are awaken.

This also helps us to adjust difficulty of each level depending on the level. Lower levels will longer and more scatter mode and steadily decreases as levels increment.

# Top Level View

## Single PLayer

1. The main initializes the programs:
   1. The player goes through the menus which use SceneBuilder as well as CSS.
2. Once the character selects “Single Player”, the character is brought to a screen where the player can choose the model and the name of the character.
   1. The model and name are then set with the Character.java class
3. LevelSelect.java then initializes the first story level for the player.
4. LevelNumber.java displays a speech box, reads the script and then displays it as character and dialogue.
   1. The script is received from Story.java, and the character is received from Character.java
5. Once the player reaches the end of the story, then LevelSelect.java initializes the first gameplay level.
6. FirstNumberStage.java generates a map, contains the user movements and control as well as the AI.
   1. The map is generated from using the level design from levelData.java
7. Once the player has finished the gameplay level, then LevelSelect.java changes the scene to the Victory Screen
   1. Or it sets the scene to the Fail Screen if the player loses all their life or runs out of time.
8. After the Victory Screen, the player is sent to the Upgrade screen, which they can change their character attributes depending on the points they gained.
   1. If attributes are changed, the incrementors or de-incrementors are called from the character class.
9. Repeat until credits.

## multiplayer

1. The main initializes the programs:
   1. The player goes through the menus which use SceneBuilder as well as CSS.
2. Once the player selects “Multiplayer”, the character is brought to a screen where they can select the model for the user and the enemy users.
   1. The model of the user are then set with the Character.java class
   2. The model of the user enemy are then set with the Enemy.java class
3. The next button in the character selected initializes the map for the multiplayer map
   1. The map initialization is called from the MultiplayerStage.java class.
4. MultiplayerStage.java generates a map, contains all user movements and control, as well as two AIs.
   1. The map is generated from using the level design in levelData.java
5. Once the player has finished the gameplay level, then LevelSelect.java changes the scene to the Victory Screen.
6. At the Victory screen the player can choose to retry, or go back to the menu.

# Issues during development

1. AI :
   1. Since we had absolutely no prior knowledge in java as well as AI, it was incredibly difficult to even begin creating algorithms for it.
   2. It was extremely time-consuming to debug AI as there are more than 300 lines of code for each AI and it was not fun to print out a lot of things and check for errors and miscalculations when just one tiny error occurs.
2. Map and Control :
   1. Initially an array builds the map, where every time there was a ‘1’ inside our level design array we would create Node so we could detect whether the user could move. The issue was that with our code for collision detection, which was to stop the user if they encountered a wall, we discovered that after every edge of a wall, the user would encounter a new wall which froze the character. The solution to this was to make the user model smaller and increase the distance for the collision detection so that there’s a small gap between the character and the wall so that the character’s bounds cannot intersect with the walls bounds. This also helped with moving the character into gaps, where the character would have to move diagonally to achieve this.

# Suitability of the tools

Java is a beginner-friendly language and has already proven to be easier to use than C++ which we had previously used. Java also has the 2nd largest StackOverflow Community, which is very useful since throughout this project, since although we were given a brief rundown of Java, JavaFX and SceneBuilder, we had to go through StackOverflow to solve a lot of problems regarding bugs/issues and functionalities we wanted to implement into our game.

JavaFX is also based on Java language so won’t need to learn how to use Unity and C# or any other game engine since our game is simplistic.

# OO-Design, Cohesion and Coupling

Regarding OO-Design, we did well overall, however we definitely could have made the code more reusable. For our character class, it focuses on only setting and getting attributes of a character and nothing more which would be high cohesion, or our enemy class which only returns a Node of its type. Meanwhile some classes go overboard with what is inside them, our actual levels are a cluster of code, there are a lot of code inside that relies on the programmer changing a lot of little things before they work for another level, however the actual code inside stays within itself, there is little influence onto the next level aside from changes to the character class. Granted the things within our game level’s codes could be much better, we could’ve made a class with different methods instead of stuffing it all inside a single class.

# Software Development Methodology

Initially we were going for a more incremental development method, where one person would work on the transition between menus, whereas the other person would work on the gameplay. Once we reached a point where both sections are good enough to function with the other we combined them then tested it, then we did the same thing over again. With the game nearing completion, we decided to go through an iterative development, where we would review past code and find that they were inefficient or were did not serve a purpose in our code. During this process, we would find new things we wanted to implement to make the game look/function better, so we would continuously review it and suggest new ideas on how to fix the issue then move on.

# Game Design Experience

Designing a game was much more exciting than many assignments we’ve had in the past, we had a lot of control over the functionality and the aesthetics of our game which made it even better as both of us had a lot of ideas regarding what to do with the game. Even though it was fun for the most part, it was challenging in a lot of ways. Firstly, it was starting the game, both of us had little to no experience designing a game, and coding with Java, we had no idea how to start. Secondly, our progress with the game was entirely up to us, there was no strict timetable to which we had to keep up with, the timetable that we set for ourselves was very loosely followed. Thirdly, we realized near the end that although we had planned out to have several more implementations, we couldn’t due to time constraints. Being in an environment where many other students are developing the same type of game helped as many people had similar issues regarding controls, AI, displays, etc.

# Suggestion for Improvement

1. Controls and Movement:
   1. The movement of the player is rather clunky; it requires the player to hold several keys to move through gaps. Through talking to others, using a queue in Java may have helped so your character could go through the next available gap when queued.
   2. The animation of the player doesn’t always display the key the player presses; this is due to pressing several keys at the same time so that the program understands that you want to move one direction but not the others.
2. Level Designs and UI
   1. We initially wanted to randomize the map by either making several maps and cycling the decision with rand() or we wanted certain blocks to be able to change into walls so that the game is still winnable.
   2. Design-wise, the aesthetics of the game could’ve been better with certain areas, such as the score and life bars as well as the GAMEOVER screen.
3. Time Management and Decisiveness
   1. Throughout the process we weren’t always on track with our set schedule, this was either due to some sections in game being more difficult to implement or that there were some consistent bugs that we needed to fix.
   2. When both of us had gotten used to the code and how to program with Java and JavaFX, we started throwing suggestions which we both agreed would be good but didn’t consider the time constraint and ended up abandoning it and ended up wasting time.
4. Code Quality
   1. In terms of OO-Design, we did well, but code quality in some sections of the code could be improved.
      1. Our gameplay levels were VERY similar in terms of code; we could have made a method and class to reuse it without pasting in the whole lot of code into another gameplay level.
      2. Our story levels had similar code as well, granted some of it was different as one of them had events with choices which could not be applied to others but the rest was reapply-able to other story levels.
   2. Commenting quality could be better, where some just state what the function is doing, whereas other don’t and write an actual purpose for the code, or what it is.

# C:\Users\pjoe652\AppData\Local\Microsoft\Windows\INetCache\Content.Word\story.pngAppendix

