SOFTWARE MANUAL

— *New Hearth*

Team No.4

May 2020

Table of Contents

1. INTRODUCTION

Our project is to develop a story-based interactive audio game for those with physical challenges. Our target user was decided to be those with visual impairment and hopefully can provide them with joyful experience through this adventure.

2. DEFINITION OF PROJECT TERMINOLOGIES

a) Watson Assistant

A conversational AI platform that helps to provide customers with fast, straightforward, accurate answers to their question, which builds, designs and deploys specialized virtual assistants or chatbots on any application.

b) Text to Speech

A cloud service that enables users to convert written text into natural-sounding audio in a variety of languages and voices. With the IBM Text to Speech service, developers can use SSML to control the synthesis of the text with all supported languages.

c) Speech to Text

A cloud-native solution that uses deep-learning AI algorithms to apply knowledge about grammar, language structure and audio signal composition to create customizable speech recognition for optimal text description.

d) SSML (Speech Synthesis Markup Language)

An XML-based markup language provides annotations of text for speech-synthesis applications. SSML provides developers of speech applications with a standard way to control aspects of the synthesis process by enabling them to specify pronunciation, volume, pitch, speed, and other attributes via markup. It is used in text to speech technique.

e) Combat System

A system that used to handle combat data, provides users a way to fight against the enemy using voice-commands.

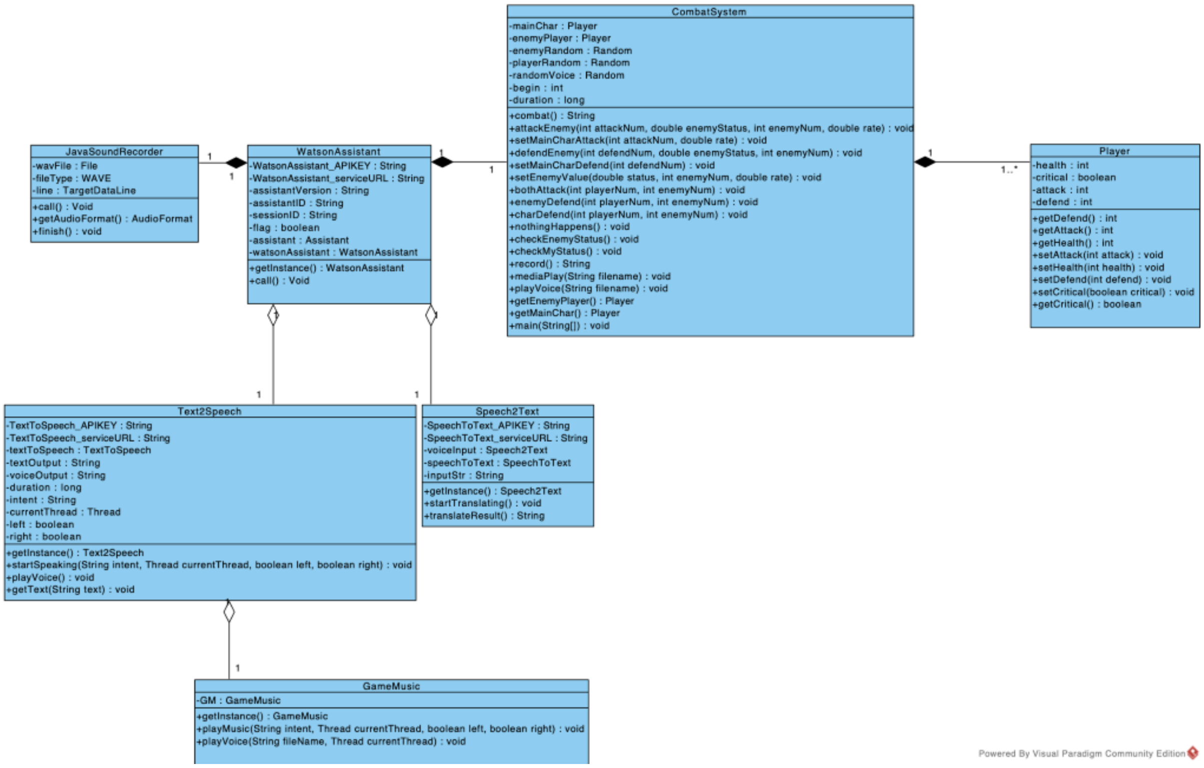
f) Multi-Threaded Programming

The ability of a program or an operating system process to manage its use by more than one request at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.

3. HIGH-LEVEL ARCHITECTURE DESIGN

The program is created uses Java alongside the Watson assistant API and text to speech/speech to text API's in order to create a fully functioning voice-based game. All storylines and conversations are stored online except the combat system. The combat system is designed using Java for data manipulation and all sound effect using in the game are in local folders as well.

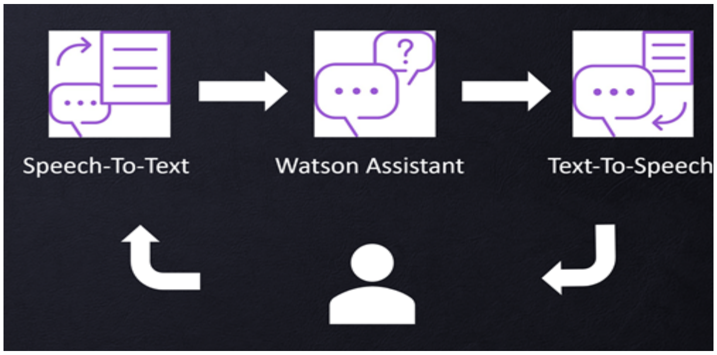
a) Class Diagram Overview


CombatSystem
-mainChar : Player
-enemyPlayer : Player
-enemyRandom : Random
-playerRandom : Random
-randomVoice : Random
-begin : int
-duration : long
+combat() : String
+attackEnemy(int attackNum, double enemyStatus, int enemyNum, double rate) : void
+setMainCharAttack(int attackNum, double rate) : void
+defendEnemy(int defendNum, double enemyStatus, int enemyNum) : void
+setMainCharDefend(int defendNum) : void
+setEnemyValue(double status, int enemyNum, double rate) : void
+bothAttack(int playerNum, int enemyNum) : void
+enemyDefend(int playerNum, int enemyNum) : void
+charDefend(int playerNum, int enemyNum) : void
+nothingHappens() : void
+checkEnemyStatus() : void
+checkMyStatus() : void
+record() : String
+mediaPlay(String filename) : void
+playVoice(String filename) : void
+getEnemyPlayer() : Player
+getMainChar() : Player
+main(String[]) : void
JavaSoundRecorder
-wavFile : File
-fileType : WAVE
-line : TargetDataLine
+call() : Void
+getAudioFormat() : AudioFormat
+finish() : void
WatsonAssistant
-WatsonAssistant_APIKEY : String
-WatsonAssistant_serviceURL : String
-assistantVersion : String
-assistantID : String
-sessionID : String
-flag : boolean
-assistant : Assistant
-watsonAssistant : WatsonAssistant
+getInstance() : WatsonAssistant
+call() : Void
Player
-health : int
-critical : boolean
-attack : int
-defend : int
+getDefend() : int
+getAttack() : int
+getHealth() : int
+setAttack(int attack) : void
+setHealth(int health) : void
+setDefend(int defend) : void
+setCritical(boolean critical) : void
+getCritical() : boolean
1
1..*
Text2Speech
-TextToSpeech_APIKEY : String
-TextToSpeech_serviceURL : String
-textToSpeech : TextToSpeech
-textOutput : String
-voiceOutput : String
-duration : long
-intent : String
-currentThread : Thread
-left : boolean
-right : boolean
+getInstance() : Text2Speech
+startSpeaking(String intent, Thread currentThread, boolean left, boolean right) : void
+playVoice() : void
+getText(String text) : void
Speech2Text
-SpeechToText_APIKEY : String
-SpeechToText_serviceURL : String
-voiceInput : Speech2Text
-speechToText : SpeechToText
-inputStr : String
+getInstance() : Speech2Text
+startTranslating() : void
+translateResult() : String
GameMusic
-GM : GameMusic
+getInstance() : GameMusic
+playMusic(String intent, Thread currentThread, boolean left, boolean right) : void
+playVoice(String fileName, Thread currentThread) : void
Powered By Visual Paradigm Community Edition

b) High-level Structure

- **Chatbot-like Framework**

  The general structure of how the program works is as followed. Firstly, the player is asked for a command/ what to do next. They respond by recording their voice saying a command. This is then fed into speech to text which converts the audio file into words. These are then sent to Watson assistant via java which in turn processes their request and then responds back to the java program. The program then uses Text-to-Speech to translate the response from WA.



c) Combat System

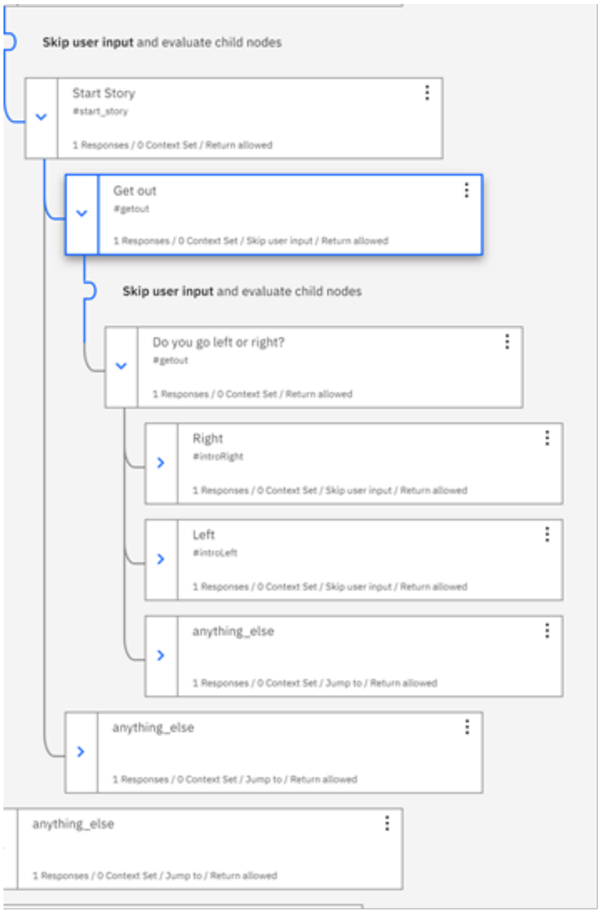
When certain intent related to combat system is detected, the java will directly jump into combat system. After certain winning or losing condition is achieved, the program will continue looping and ask response from Watson Assistant.

4. PROJECT SPECIFIC SOFTWARE

a) Watson Assistant

Watson assistant is the main brain behind the program, controlling whether the players input was valid and if so where the player goes next based off the commands they said.

Watson assistant works based off a dialog tree structure. Each node in this tree structure can have many things one of which being an intent. An intent is a set of key words and user examples that you would want to trigger that node. When a valid input comes, the program will repsond by checking each of the child nodes intents and seeing if the commands input matches that intent. If so, it returns the dialog in that node and progresses the story. Otherwise it returns an error message and repeats the last question Watson assistant returned.



Skip user input and evaluate child nodes

Start Story
#start_story
1 Responses / 0 Context Set / Return allowed

Get out
#getout
1 Responses / 0 Context Set / Skip user input / Return allowed

Skip user input and evaluate child nodes

Do you go left or right?
#getout
1 Responses / 0 Context Set / Return allowed

Right
#introRight
1 Responses / 0 Context Set / Skip user input / Return allowed

Left
#introLeft
1 Responses / 0 Context Set / Skip user input / Return allowed

anything_else
1 Responses / 0 Context Set / Jump to / Return allowed

anything_else
1 Responses / 0 Context Set / Jump to / Return allowed

anything_else
1 Responses / 0 Context Set / Jump to / Return allowed

For example, on the left, arbitrary nodes can exist in our program. Here the player is presented a large amount of text in get out and then they are asked if they want to go left, or right?

Watson assistant then passes control back to Java as the local program waits for an input. Once the player has said a command it is then converted and passed back to Watson assistant. This will then compare the words inputted to the intents #introRight and #introLeft which will either trigger if the input was in that intent or not.

If the player says right the #introRight intent will be recognised and so the Right node will be read, and the program will progress through that child node.

If none of the children's intents are matched the program will go to the anything else node which will respond with an error message and jump back to left or right?

b) Text-to-Speech

To allow sight impaired people to be able to listen to the game, we obviously had to implement Text-to-Speech. This makes the computer read out whatever we want it to, so that they can hear the story. We could also make the synthesised voice sound more interesting by using SSML, Speech Synthesis Mark-up Language. This allowed us to change the speed, pitch and put breaks into the text, making the voice sound a lot more natural and kinder on the ears. While entering the text into Watson Assistant, to test that the section of text would sound good with the synthesised voice, we used an implementation by Bluemix. We input the text and put in varying types of SSML to see how natural we could make the text, then we put it back into Watson Assistant once complete.

c) Speech-to-Text

As sight impaired people are less likely to want to use a keyboard or any kind of visual input, we had to make the game focused around being able to use your voice to control the game. This meant we needed to implement Speech-to-Text. We would ask for a command, prompting them to use their voice as input to control the characters actions. Again, we had to make sure this worked well, so we had to test it using an implementation by Bluemix, we tested how well Watson's Speech-to-Text worked and how well we could implement it into our game. More details on the testing will be talked about later in the document.

5. DATA STORAGE TECHNIQUES

The data are stored separately into two different places, online and local folders. For the storyline in the project, it is stored online on IBM Watson Assistant by using multiple nodes and tree structures. Each time when response is needed from Watson Assistant, users can simply send text through Watson Assistant API and wait for it.

Considering Watson Assistant only return plain text without any sound effect, to increase the gaming experience, sound effect is added in the local platform, stored in "resource" folder. That includes the music inside combat system as well. Developers can easily put the music they want to add inside "resource" folder. If it's a music used in Watson Assistant, code related to it can be added in class "GameMusic", and if it's a music used in Combat system, code related to it need to be added in class "CombatSystem".

To minimise the network delay of the project caused by Text to Speech and Speech to Text techniques, the audio in combat system is pre-processed and stored in "Combat system" inside "resource" folder.

6. EXTERANL LIBRARY / DEPENDENCY

- **"com.ibm.watson"**
  Included in maven repository, it wrapped up the classes and libraries needed for successful implementation with Watson Assistant**.**
- **"org.openjfx"**
  Included in maven repository, it used for the successful implementation of the JavaFx.
- **"junit"**
  Included in maven repository, it used for successful implementation of the unit testing for combat system

7. MAINTENANCE

a) Coding convention

Since our project is primarily developed based on Java, the coding convention matches oriented-object concepts, where we have focused on components that the user perceives, with objects as the basic unit. The code pattern is implemented according to the standards of OOP programming, such as naming convention, variable declaration, error handling, etc.

b) Comment

Comments are an integral part of any program. They help the person reading the code better understand the intent and functionality of the program. In our program, we added comment according to the functionalities of classes, methods or even some variables to ensure the code’s readability.

Javadoc is another type of comment we have used in our project. It is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format. It parses the declarations as documentation in a set of source file describing classes, methods, constructors and fields. Javadoc Comments allow us to see the method and the return value of that method on hover. This can save a lot of time when coding. Also, it allows other team members to read method and get the general idea from it.

c) Design pattern

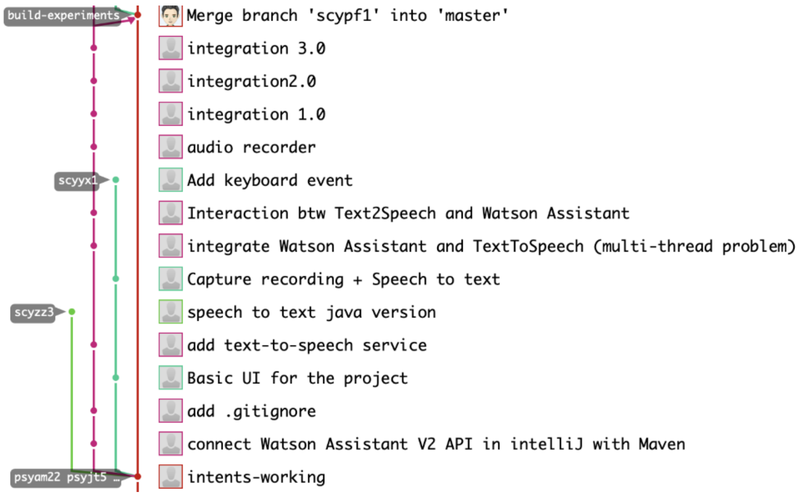
A design pattern is a well-described solution to a common software problem, which saves us a lot of time. Using design patterns promotes reusability that leads to more robust and highly maintainable code. Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily. Generally, our team has used Singleton design pattern and MVC design pattern in our project.

Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the Java virtual machine. It seems to be a very simple design pattern but when it comes to implementation, it comes with a lot of implementation concerns. In our project, we have applied it to class Watson Assistant, Text to Speech, Speech to Text and Music, since these classes only need one instantiation and random use of duplication may cause program errors.

MVC stands for Model-View-Controller. The idea behind MVC pattern is a very clear separation between domain objects which represents real-world entities and the presentation layer we see on the screen. Domain objects should be completely independent and should work without a View layer as well. In our project, Model refers to WA, TTS and SST, which carries data; View represents the User Interface; Controller is class “GameController”, which deals with user input and software output events. The Model View Controller design pattern enables us to isolate the concerns and makes our application's code easier to test and maintain.

d) Version control

Version control is an important component of software configuration management, especially when development gets more complex and there's a larger need to manage multiple versions of entire products. Version control is important for all code, files, and assets that multiple team members will collaborate on. Our team has used GitLab to manage our code, by which we can keep track of changes and keep every team member working off the latest version. We have set the code repository into many branches, where ‘Master’ is the main branch and is used to store the latest version of our code, while the rest branches contain each member’s trial. Every time one of our members finishes developing his task, we will have a discussion and give a evaluation before merge the work to ‘Master’. Also, if a mistake is made, we can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.



```
Merge branch 'scypf1' into 'master'
integration 3.0
integration2.0
integration 1.0
audio recorder
Add keyboard event
Interaction btw Text2Speech and Watson Assistant
integrate Watson Assistant and TextToSpeech (multi-thread problem)
Capture recording + Speech to text
speech to text java version
add text-to-speech service
Basic UI for the project
add .gitignore
connect Watson Assistant V2 API in intelliJ with Maven
intents-working
```

8. TESTING APPROACHES

a) Unit testing for Combat system

As combat system including some data manipulation, such as some getter and setter functions, unit testing is required to know whether each small unit is error free. Unit testing helps developers to decide that individual units of the program are working as per requirements and are error free.

A unit testing framework of basic combat system is provided inside the program, and inside “test” folder, developers can easily perform unit testing by clicking the “run” button of that unit test.

b) Integration Testing for Combat system

Integration testing is needed to check if the units if integrated together would also work without errors. As integration testing is mostly voice-based, our testing approaches need to check both object’s data update and correct voice feedback.

All situations that required to be tested are listed inside the document called “Integration Testing for Combat System”. The testers need to ensure all situation are being tested without any missing voice or wrong data by directly run the class called “CombatSystem”, which contains a main function for easy testing.

c) Test Plan for Watson Assistant

The testing plan for Watson assistant was split into two main parts: Testing of the intents and play testing of the story.

- *Testing of the Intents*

  The largest aspect of our project was the player inputting commands into the game by voice. This meant that if this part performed poorly by not recognizing the commands or by crashing when a wrong word was inputted then the whole game and its playability would suffer. Therefore, to try and reduce this risk as much as possible we tested each of our intents and the user examples they used.

As mentioned above, to do this we used an excel spreadsheet to test each of our intents several times (20) recording the accuracy of them as well. If their accuracy was lower than 80%, we decided that they would need some changing or often tweaking in order to boost this rating. Some of the fixes we made were easy like adding some more user examples to cover speech to text going wrong but other times like for the intent #hallway it had an accuracy of 15% which was due to it recognizing “hallway” as “holway” or ”holy” so we had to completely change the location area to something that was more recognizable in order to improve the overall game experience.

- *Play Testing*
  Another important aspect of the project was the story itself and if it flowed properly and didn’t crash. With Watson being the main brain behind the program it was important to test that the story flowed within this before being connected to the java side of the program. To test this, we simply play tested the story after every new planet/ chunk of the story was added, testing every path the player could go and testing wrong commands as well to make sure the story wouldn’t break.

d) System Testing

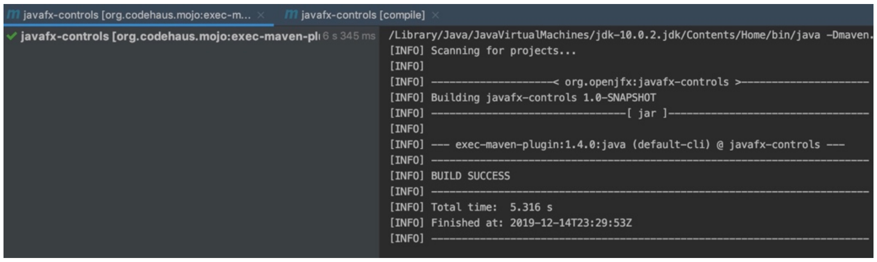
After each unit is finished testing, system testing is used to test the product. After combining the code in local platform together with Watson Assistant, we tested the storyline ensuring our demo works smoothly.

9. PROJECT INSTALLATION AND PUBLICATION

a) Maven Framework Deployment

Our group uses the maven to import external dependencies such as JUnit, JavaFX, and IBM Watson assistant. Therefore, after downloading from the GitLab, the first task is to use maven to compile and install these external dependencies. These dependencies will appear in the ‘External Libraries’ list after successfully importing them.

Step 1: Maven -> Lifecycle -> compile
Step 2: import changes
Step 3: Maven -> Plugins -> exec -> exec: java
If see the following similar result, it means you run program with Maven Successfully.


javafx-controls [org.codehaus.mojo:exec-m...
javafx-controls [compile]
✓ javafx-controls [org.codehaus.mojo:exec-maven-pl 6 s 345 ms

```
/Library/Java/JavaVirtualMachines/jdk-10.0.2.jdk/Contents/Home/bin/java -Dmaven.
[INFO] Scanning for projects...
[INFO]
[INFO] -------------------< org.openjfx:javafx-controls >--------------------
[INFO] Building javafx-controls 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec-maven-plugin:1.4.0:java (default-cli) @ javafx-controls ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  5.316 s
[INFO] Finished at: 2019-12-14T23:29:53Z
[INFO] ------------------------------------------------------------------------
```

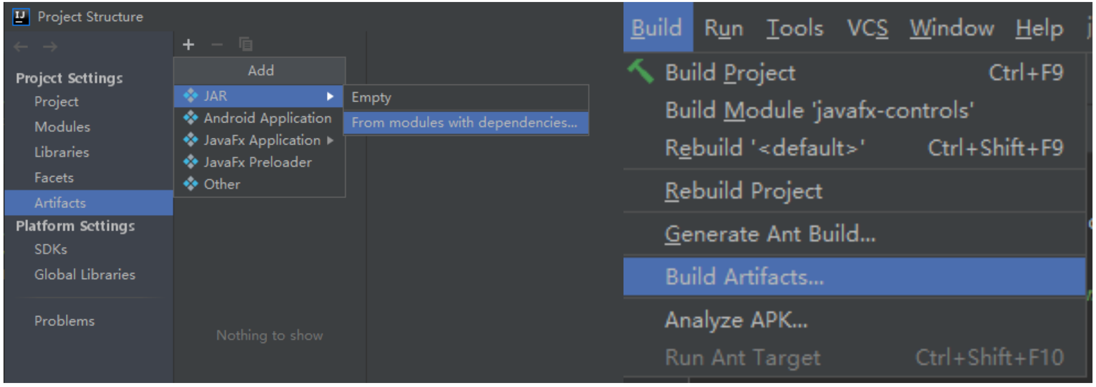
IDEA installation with Maven only supports for the Java 10 version. If the game cannot run successfully, please check the java version or try our execution version directly.

b) Software Export

The execution version of the game needs to be updated once the program is changed.

Step 1: Export the program into .jar file

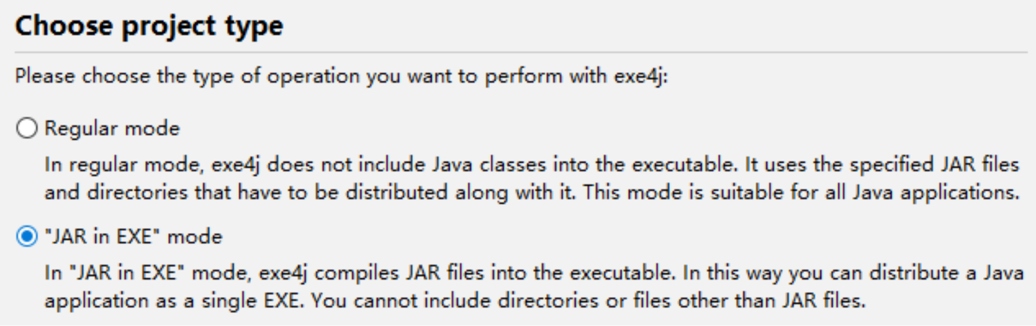
- Firstly, click the File -> Project structure -> Artifacts -> Add to add the configuration of the project and then choose the Main.java as the main class.



- After that, click the Build -> Build Artifacts to build up the current artifacts and the .jar file will be exported as the same time.

Step 2: build up the .exe file with exe4j

- First, download the exe4j tools, then choose the 'JAR in EXE' mode



Choose project type

Please choose the type of operation you want to perform with exe4j:

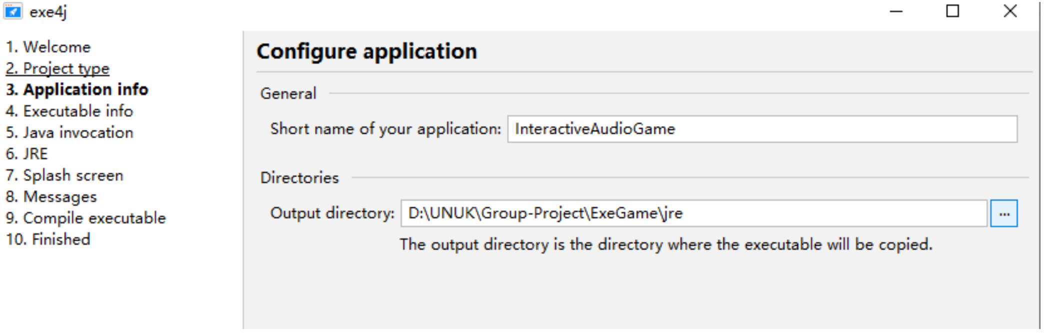
○ Regular mode

In regular mode, exe4j does not include Java classes into the executable. It uses the specified JAR files and directories that have to be distributed along with it. This mode is suitable for all Java applications.

● "JAR in EXE" mode

In "JAR in EXE" mode, exe4j compiles JAR files into the executable. In this way you can distribute a Java application as a single EXE. You cannot include directories or files other than JAR files.

- After that, give the project a name and configure the output directory. Additionally, in this directory, put the java 10 package inside for later usage.



Configure application

General

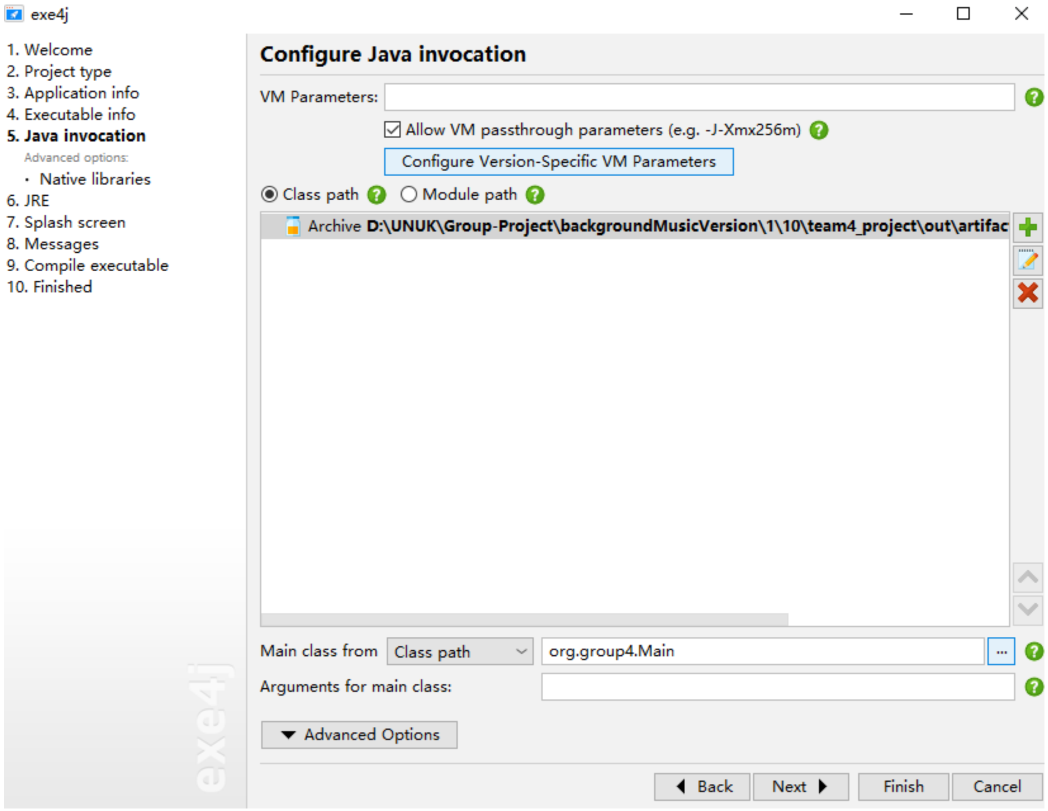
Short name of your application: InteractiveAudioGame

Directories

Output directory: D:\UNUK\Group-Project\ExeGame\jre

The output directory is the directory where the executable will be copied.

- Then, click the GUI application choice, import the .jar file and then choose the Main.java as the main class.



Configure Java invocation

VM Parameters:

☑ Allow VM passthrough parameters (e.g. -J-Xmx256m)

Configure Version-Specific VM Parameters

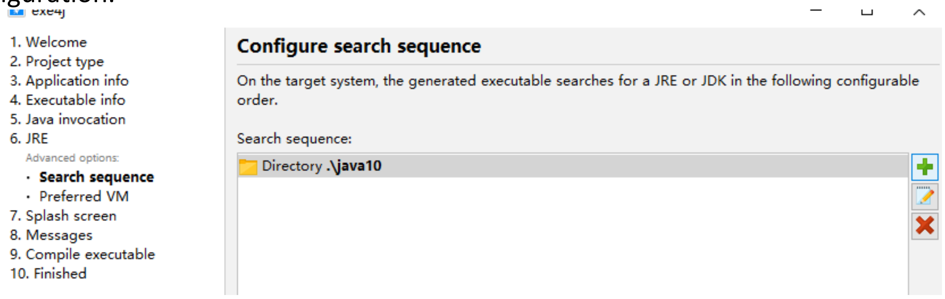
● Class path ○ Module path

Archive **D:\UNUK\Group-Project\backgroundMusicVersion\1\10\team4_project\out\artifac**

Main class from: Class path — org.group4.Main

Arguments for main class:

- The most important step is to import the java 10 directory and delete the other default configuration.



Configure search sequence

On the target system, the generated executable searches for a JRE or JDK in the following configurable order.

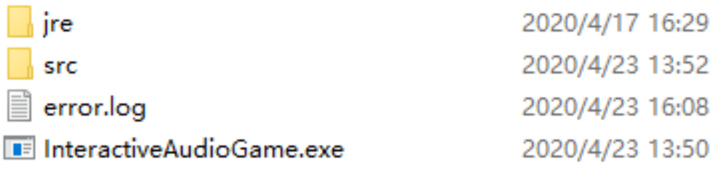
Search sequence:

Directory **.\java10**

- Finally, the executable version of the project will be successfully exported.

c) Launching the Game

Double click the 'InteractiveAudioGame.exe' directly and wait for the game loading for a few seconds and then it will run successfully.



jre	2020/4/17 16:29
src	2020/4/23 13:52
error.log	2020/4/23 16:08
InteractiveAudioGame.exe	2020/4/23 13:50