**TEMASEK POLYTECHNIC**

**SCHOOL OF INFORMATICS & IT**

**DIPLOMA IN GAME DESIGN & DEVELOPMENT**

**AY2023/2024 APRIL SEMESTER (LEVEL 3) TERM A**

**GAME AI (CGE3E01)**

**Assignment 2 – SUBMISSION DOCUMENT (10%)**

**Name: Ian Wong Chong Shen Total Marks / 2 =**

**Class: P03**

**Question 1 (Not graded)**

Give a brief description of your robot’s behaviour after adding, changing, or removing any behaviours with respect to the behaviour described in your Assignment 1. You should describe the robot’s behaviour in point format, for easy identification of behaviours.

*This question is not graded but will be used in the Runtime Behaviour section of your Requirement 1 marking rubric.*

* The Potion Maker Bot’s behaviour stays roughly the same. However, there are a few differences.
* While in Idle, the bot walks between locations randomly. While walking around, if the player comes near the bot, it stops in its tracks.
* The Approached behaviour stays roughly the same as well, with the only difference being that the bot also moves to the counter.
* The Attending behaviour stays roughly the same.
* The Potion Inquiry behaviour stays roughly the same.
* The Transaction behaviour has been slightly altered. Instead of going back to the Attending tree, it instead treats it as though the player has left. This was mainly due to constraints and how the Attending tree was circularly defined, so there wasn’t much of a way I could think of to have the Transaction tree go back to the Attending tree.
* The Studying behaviour stays mostly the same, just that the bot must move to the studying location first before doing anything in it.
* The Check Component behaviour stays mostly the same, but there is one main alteration. If the player chooses to not go through with the brewing process, it instead treats it as though they left.
* Brewing behaviour stays mostly the same, although the bot must move to the brewing station first.
* Cleaning behaviour has been changed in how it is called. Instead of going into the Cleaning tree if the boolean `dirty` is true after the player leaves, the bot instead cleans up each time after brewing to simplify it.
* Failed behaviour stays the same, except once again if the player chooses to abandon the brewing process it treats it as though they left. Additionally, if the player chooses to continue, the brewing timer is also halved and it is a guaranteed success on the re-attempt.
* Requesting behaviour stays mostly the same, except if the player chooses not to give it treats them as though they left.

**Question 2 (10 marks)**

The factors in the table below represent steps in the design and implementation of a robot’s behaviour using a finite state machine or a behaviour tree approach.

For each factor, write which approach you prefer (FSM or BT), and then give THREE reasons for your preference.

Your reasons must be in **point form**.

|  |  |  |
| --- | --- | --- |
| **Factor** | **FSM/BT** | **Reason(s)** |
| **MODELING the robot’s behaviour.**  **Breaking the robot’s behaviour down into individual actions and conditions.** | **FSM** | * FSM are easier to model or think up of since you only need to really think of two things: State and Conditions, while you need to think of things like sequence nodes for BTs. * Because of the simplicity, you can think up of a FSM model quite a bit quicker. * States in an FSM can easily be added or removed. This can also be done for BTs, but since they are hierarchical it might also impact other elements of them. |
| **DESIGNING the robot’s behaviour, e.g., creating the structure of the FSM or BT.**  **Deciding how the actions and conditions define the robot’s behaviour, including includes iterating the design, i.e., making changes to the FSM or BT structure when you implement it.** | **FSM** | * Designing diagrams for FSMs, I feel like I get more freedom in organization than designing diagrams for BTs. * When organised well, FSMs can be easier to follow as they are more linearly designed with each state transition being clear. * At the moment, when I think of the kinds of bots I implement into my games, they are usually simpler. Therefore, FSMs are better for those simpler bots and behaviours. |
| **IMPLEMENTING the robot’s behaviour.**  **Writing the actions and conditions, and other functionality. Note; this is obviously dependent on the BT plugin you choose to use. Just write about your experience.** | **BT** | * During development, bots can definitely get more and more complicated. Though I am less familiar with BTs, I can see their use in creating complex behaviour and keeping a readable workflow. Without reference to my FSM diagram, I started to find it easier to read the Panda BT file as more states were being added than I did looking at comments from my FSM code. * Since FSMs have more predefined states and transitions, it’s more difficult to implement more dynamic behaviours. BTs have less of a problem with this, and I found that out over the course of the assignment. * While I used Panda BT, which doesn’t have a GUI editor, I would say that it is easier to look at a GUI editor than it is to look at FSM code. |

**Question 3 (5 marks)**

In the tables below, write FIVE aspects of your chosen behaviour tree plugin that you LIKE, and FIVE aspects that you DISLIKE.

Give a reason for each aspect that you like / dislike.

**Chosen Plugin:** Panda BT

**LIKE**

|  |  |
| --- | --- |
| **Aspect** | **Reason(s)** |
| **Programming-Based** | Weirdly enough, I tend to prefer programming things over using built-in actions. Panda BT in that sense felt more comfortable for me to use. |
| **Readable BT** | The idea of not relying on a GUI editor seemed pretty bad at first, but the text file that contained the BT proved surprisingly readable and easy to follow. |
| **Easily Implemented Tasks** | Tasks were easy enough to implement, as they were basically just functions with the [Task] attribute. |
| **Learning** | Since Panda BT is more coding-reliant, it was much easier for me to pick up than other plugins with a lot of built-in features. |
| **Flexible** | Because Panda BT tasks are written as functions, creating unique behaviours and customisation were far easier to do. |

**DISLIKE**

|  |  |
| --- | --- |
| **Aspect** | **Reason(s)** |
| **Lack of Online Resources** | During the assessment, I found it difficult to find proper concise resources for the exact problems I had, which meant that most of the time spent was on trying to solve simpler issues. |
| **Maintenance** | While easier to maintain than FSMs, I can definitely see why Panda BT might be harder to maintain than if you were to use other plugins, as the reliance on scripting means you would have to go through the process of refactoring. |
| **Requirement of Programming Proficiency** | While not that much of an issue for me, if I were to work on projects in a group this could prove to be a problem. If I wanted someone else to help with the behaviours, and that person happened to not be proficient or confident in their programming skills, it would prove challenging for them to use Panda BT. |
| **Readability for Others** | As stated earlier, it proved readable enough to me. However, when discussing my implementation with others, those who preferred using BTs with GUI editors were very obviously confused with Panda BT. Some students tried Panda BT but couldn’t understand it without that visual touch. Again, this could prove challenging in group projects. |
| **Debugging** | I found it pretty difficult to debug and solve a lot of errors that came from Panda BT. This was not helped by my unfamiliarity with it. |

**Question 4 (5 marks)**

Write a brief reflection about Assignment 2.

I do feel more confident when it comes to future projects, as I have been introduced to more ways to make a bot within my game. That being said, my implementation in this assignment and how I went about it is not my proudest. While I prefer using Panda BT over the other plugin options presented, it was not easy to learn how to use it effectively. I do not think that I was able to use it to the level I should have been able to.

At many parts of the implementation, I had to change around how things worked in the previous Finite State Machine implementation, as I was unable to implement it in the same way using Panda BT. The lack of documentation and proper help with the plugin did little to help this. For example, many trees that were supposed to go back into previous trees based on specific player input instead just restart the agent and force the player to engage the potion maker again. The main examples of this is going back as an option in the Check Component tree, which in the Finite State Machine implementation was supposed to bring the player back to the Attending State.

I also had to do alternate, likely worse ways to repeat certain tasks. I tried using the repeat node, but many times it did not work for me and instead I had to use coroutines to repeat an action. That is why when you look at the PotionMakerClass.cs script, you would see a lot of the same general coroutines and functions just to do this.

The worst part of my implementation was definitely the cleanliness and overall skill shown. I was very much focused just on getting the implementation to work, that things like refactoring were not something I considered, even if I would normally refactor my code. This meant that everything was stored in one script, which means that it might not be the most readable. The best I did was making use of regions in order to be able to collapse parts of my code and organise them.

That being said, I do not think I did too poorly for this implementation. If anything, I am a bit more motivated to begin using Behaviour Trees and Panda BT instead of Finite State Machines for my upcoming major project. I do think it was important to go over Behaviour Trees for this subject, especially since they are very widely used and enable us to create more maintainable and complex A.I. for our games.