Sorbonne Université
Master ANDROIDE

ISG: Project SPY

Students :

Qingyuan Yao, Ruohui Hu, Xinyu Huang

Teachers :
Amel Yessad, Mathieu Muratet, Thibault Carron

January 30, 2023

Contents

1 Introduction

This is a report of the changes we have made on SPY, a serious game that teaches the basic notions of programming. We have tried multiple aspects to introduce new ideas to the game.

The repository of the project can be found here: `https://github.com/MRVNY/SPY`

2 Optimizations

2.1 Saving and Loading GameData

Previously, the GameData is a GameObject prefab that is created at the start of the game, it doesn't get destroyed while changing scenes and therefore is able to carry over information on the current level. Upon finishing a level, the scores and other parameters would then be stored with PlayerPrefs.

While this is a viable way to save the data. We think it would be easier for future development of the game if the GameData itself can be saved so we don't need to add and deleted PlayerPrefs every time we change a variable, and doesn't have to "physically" be in a scene. To achieve the static characteristic while being serializable, we created a static class called Global and in it we put a GameData object which has become serializable by switching its dictionaries to hashtable.

For saving, we transform the serializable data into binary data and save it in a persistent path (Application.persistentDataPath) that depends on the machine, for loading, we do the opposite.

2.2 async

Though Coroutine has been used in previous code in order to allow parallel execution, we have decided to use async from the C# library in order to do await on certain executions. This allows us to start an execution without worrying about if it will finish on time so its dependent executions could run correctly.

2.3 Settings

We have added a settings panel for personalization. For the moment, the settings only serve for changing language and deleting save data, it could be useful for other aspects such toggling accessibility features.

2.4 Bigger UI

While this is a personal preference, we do find some buttons hard to click on and some texts small to read, especially on small screens. Following the Fitts's Law, we made some UI interface bigger and more centered for easier accessibility.

3 Mechanics and Evaluation

Initially, we had thought about three other modes such as debug mode, hacker mode and adding variables as learning content, and considered adding consideration of functions, objects, and design patterns as learning content as well, due to the time constraints, we have decided to work on other aspects of the project as written below. Though we feel bad about abandoning new mechanics, we think that focusing on aspects that other groups did not think of would contribute more to this project and to this course.

Debug Mode is the ability to select when given a series of action commands, and to modify existing instruction commands. It is useful to guide the player on thinking, or misguide them.

As for evaluation, we have succeeded in separating the length of the code and the length of execution, the game would show the correspondent score based on these two criteria. However, we didn't have time to manually calculate the best scores for each level.

4 LevelEditor

The LevelEditor is the first and foremost functionality we have thought of. Despite being a common idea among students, it is crucial for level design. However, while others aim for a playable feature in the game, we prioritize the facility of implementation and flexibility for level designing.

4.1 Tilemap

Therefore, we have chosen to use the built-in Tilemap editor in Unity, which has been widely used for creating levels for 2D pixel games. The painting, erasing, and layering functionalities already works perfectly, all we need to do is to import images that represent elements from the game and paint on the correspondent layer. We have written a script which scans all the layers of the Tilemap and translate each cell into xml data.

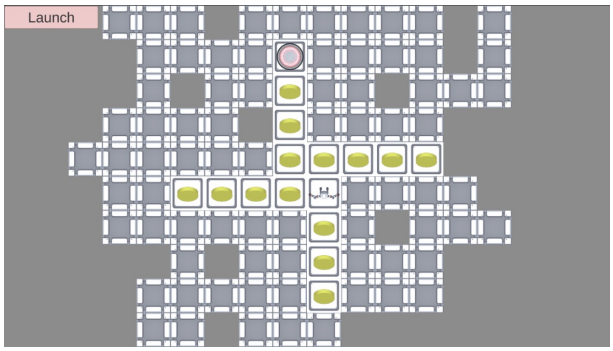


Figure 1: LevelEditor

We are perfectly aware that a xml file of the level doesn't only have a map, therefore we have created a LevelEditor class which doesn't contain any functions but variables linking to parameters of the xml file, which can be modified accordingly. The editor also allows precisely linking between consoles and doors, as well as naming the agent.

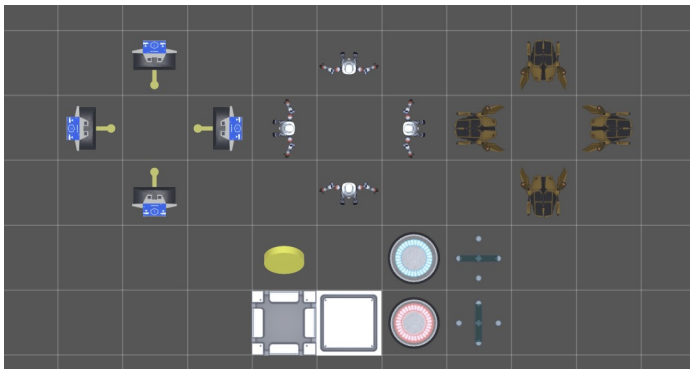


Figure 2: Tile Palette

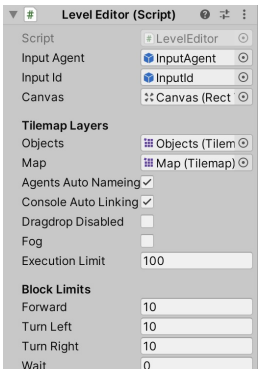


Figure 3: Parameters

4.2 ScriptEditor

As for the script in the xml file, we have decided to take advantage of the already existing interface of the block, which is already used for composing a script. However, in order to translate GameObjects into xml elements. We would need to let EditingUtility links the GameObjects in editing panel and carry them to execute panel. And iterate on the links in order to add each xml tags in order.

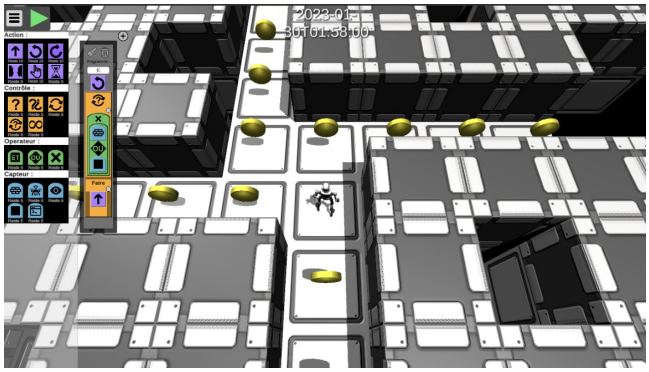


Figure 4: ScriptEditor

```
<script name="K" editMode="2" type="3">
  <action type="TurnLeft" />
  <while>
    <condition>
      <not>
        <or>
          <conditionLeft>
            <captor type="Wall" />
          </conditionLeft>
          <conditionRight>
            <captor type="FieldGate" />
          </conditionRight>
        </or>
      </not>
    </condition>
    <container>
      <action type="Forward" />
    </container>
  </while>
</script>
```

Figure 5: Generated Script

5 VisualNovel and Scenario

In order to increase interactivity and draw interest from children, we have decided to add a VisualNovel interface which would replace the DialogSystem. According to our experience, VisualNovel helps the readers to pay more attention to the dialog since they sometimes need to choose a reply according to the sentences (which is not much explored for the limited time for content creation).

5.1 Interface

Apart from the dialog box and an image of the robot, the middle of the interface switches between blank, images, input box, and choices to choose from.

The images are exactly the same from the DialogSytem, they're for tutorial purposes, we haven't added more images due to time constraints but further usage would be interesting.

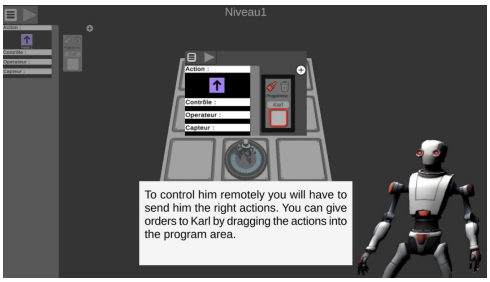


Figure 6: Image

The input box for asking player's name, which then stores it in GameData, would be used for tracing, it could also be used for other purpose such naming the robot or inputting a password, but we didn't explore further due to time constraints.

Other than letting the player have a conversation with the robot, we can also record the player's behavior from the choices they made. One of the obvious usages would be to ask the player how does he feel about the difficulty of the current level (implemented) and change the difficulty accordingly (unimplemented).

Figure 7: Ask Name

Figure 8: Ask Difficulty

5.2 json

The structure of the dialog for VisualNovel is a node which contains the text in various languages, response to choose from and the nodes they lead to, as well as image to show, camera focus position, and an action to take.

```
"askDifficulty": {
    "en" : "Wow, that was kinda hard for me to understand, what about you?",
    "fr" : "Wow, c'était un peu dur pour moi à comprendre, et toi?",
    "options" : {
        "en": {
            "askDifficulty.hard": "I had no idea what I was doing",
            "askDifficulty.medium": "I kinda undertood",
            "askDifficulty.easy": "Easy peasy"
        },
        "fr": {
            "askDifficulty.hard": "Je n'avais aucune idée de ce que je faisais",
            "askDifficulty.medium": "J'ai un peu compris",
            "askDifficulty.easy": "Facile comme bonjour"
        }
    }
},

"askDifficulty.hard" : {
    "en" : "I know right! \nLet's just try to take it slow, I'm sure we can make it!",
    "fr" : "Je sais, n'est-ce pas! \nEssayons juste de prendre le temps, je suis sûr qu'on peut y arriver!",
    "action" : "changeDiff,-1"
},
```

Figure 9: json

The action object is a mini script of what to do when finishing the current node:

- next,{node}: Set the next node
- askName: Show input box and ask the player's name
- changeDiff,{num}: Change the difficulty of the game
- ending2: Play the animation of Ending2
- ...

5.3 Storyline

The story is about a robot who doesn't know their purpose, the player helps the robot to move forward in the environment, by solving puzzles and developing basic programming knowledge. Along the way, the player and the robot find some files that document a project called SPY.

In the end they found out that they are actually subjects of the SPY project and that the environment is actually made to train the player and transfer the knowledge of the player to the robot. Once finished, the player's programming skill would be taken and given to the robot.

The player then faces two possible choices:

- Ending1: Robot sacrifices, "Please, use your programming skills to do something good for the world, and whenever you code, think of me." (implemented)
- Ending2: Player sacrifices, all saves wiped out (implemented)

However, after Ending2, the player can choose to replay the levels (skipping is possible) to reach the end again, where the player and the robot beat the system. (unimplemented)

We have managed to write some dialog with the VisualNovel, but due to time constraints, the story is far from complete.

6 LevelMap and Tree Structure

While playing the original version of the game, we found that the progression and difficulty changes were quite drastic, and we search to make a better transition of the levels.

6.1 CbKST

We have thought of CbKST In the game, we analyzed that there are 7 basic competence of programming: FOR, IF, IF/ELSE, WHILE, NOT, AND, OR. We have decided to regroup some competence together in order to simplify competence structure:

- For : know how to use the for loop
- If/IfElse : Know how to use conditions
- Op : Know how to use operators, e.g. not, and, or
- While : know how to use the while construct

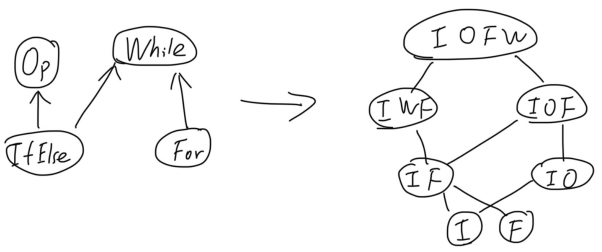


Figure 10: CbKST: Domain Model and Competence Structure

6.2 Tree, Node and Level

In order to represent the structure above, the structure of the levels are also to be reimagined. We have discarded the structure of list in the original game and opt for a Tree structure. We have created a Node class which represents a node of the competence structure. The node has the possibility to have a list of nodes that comes next. The tree starts from one node and expands to others via this link.

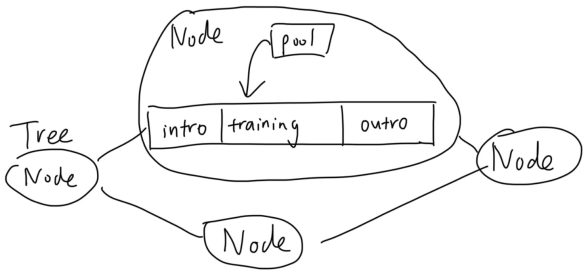


Figure 11: Node Class

In each Node, There is a sequence of levels which is represented by class Level. The levels are grouped into 3 sub-sequence:

- intro: Introduction to the competence that must be played in order
- training: a sequence of levels pulled from a pool of levels
- outro: Ending (aka. boss levels) of the competence that must be played in order

The purpose of having a training sub-sequence is so that we can adapt the levels to the current competence of the player.

6.3 LevelMap

In order to represent the competence structure to the player, we have decided to construct a map that builds the Node tree structure. We have decided to use the Tilemap again, but this time, it's a hexagonal Tilemap which allow us to build a tree structure. The current version we made accepts up to 4 branches going at the same time. However, the branches need to be merged before opening other ones and the parallel branches must have the same length.

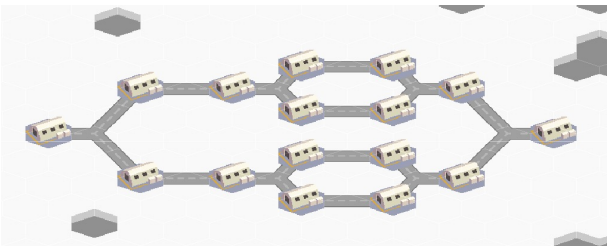


Figure 12: Maximum Branches

This has made our competence structure impossible to construct, therefore we have simplified one more time, by taking out the If_Op node, which means the Operators will be introduced after the player has understood both If and For.

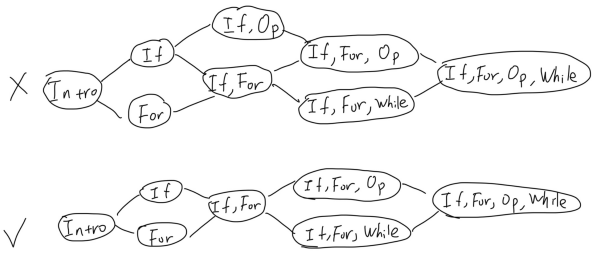


Figure 13: Limitation

As we have thought before, the last level of the outro would be a boss level, and therefore available to skip to from any previous levels of the same node.

6.4 Level Adaptation (unimplemented)

One of the reason we build the LevelMap and the tree structure is for adapting to player's skill levels for different competences. We envisioned doing this in the pool of levels of each Node. We would also like to calculate the competence each level has, by counting the actionLimits; and evaluate player's skills with the help of tracing. With information of competences for both levels and the player, we would ideally be able to calculate the best level suitable for the player with the MAGAM model. However, both the tracing and level analysis, couldn't be finished.

7 Tracing

For a Serious Game, the part "Trace" is very important for professors to know

- concepts that are poorly mastered by students.
- specific student who is having trouble with certain concepts/levels
- students' attendance

And also for a specific student, do comparison with other students with critics such as

- score obtained in each level
- quality of answer(in terms of execution length)

- time to accomplish a level (duration)
- number of level completed

7.1 Vocabulary used

A statement in XAPI is composed of four elements :

- Actor
- Verb
- Object
- Extension (optional)

The structure of can generalize by :

Actor + verb + object + Extension

The element "Actor" will be given by the player when they enter the game. Thus, three kinds of vocabulary should be defined according to functionalities that we want to implement.

quitte
demarre
recommence
commence
choisit
active
essaie
reussit
revient sur
maitrise
reinitialise

(a) Verb

serious-game SPY
score
action
level
Menu
competence_If
competence_While
competence_For
competence_Operator
donnee_personnel

(b) Object

lv
score
actions
duration
nb_lv_completed
code_length
execution_length
competence_nb

(c) Extension

7.2 Functionality implemented

In our serious-game "SPY", we have generally two kinds of trace to implement:
- Button trace : Traces will be sent on clicking a button

- **SendBeginGame** : On clicking the button "level" of the "titlescreen" scene.
- **SendQuitGame** : On clicking the button "quit" of the "titlescreen" scene.
- **SendRestart** : On clicking the button "restart" of the "Winlevel" panel.
- **SendBackMenu** : On clicking the button "menu" of the "Winlevel" panel.

- Evenement trace : Traces will be sent when an evenement happened.

- **SendLevel** : Send a statement when the player start a level.
- **SendActions** : Send a statement containing a sequence of actions that the player choose as an answer of the level.
- **WinLevel** : Send a statement when the player wins a level, containing the level, score, duration, number of level completed, length of code/execution in the extension.
- **ResetData** : Send a statement when the player wants to reset the data.

The most important functionality here is **WinLevel** because it not only tell us the level is completed by the player, it also provides a lot of important information in its extension.

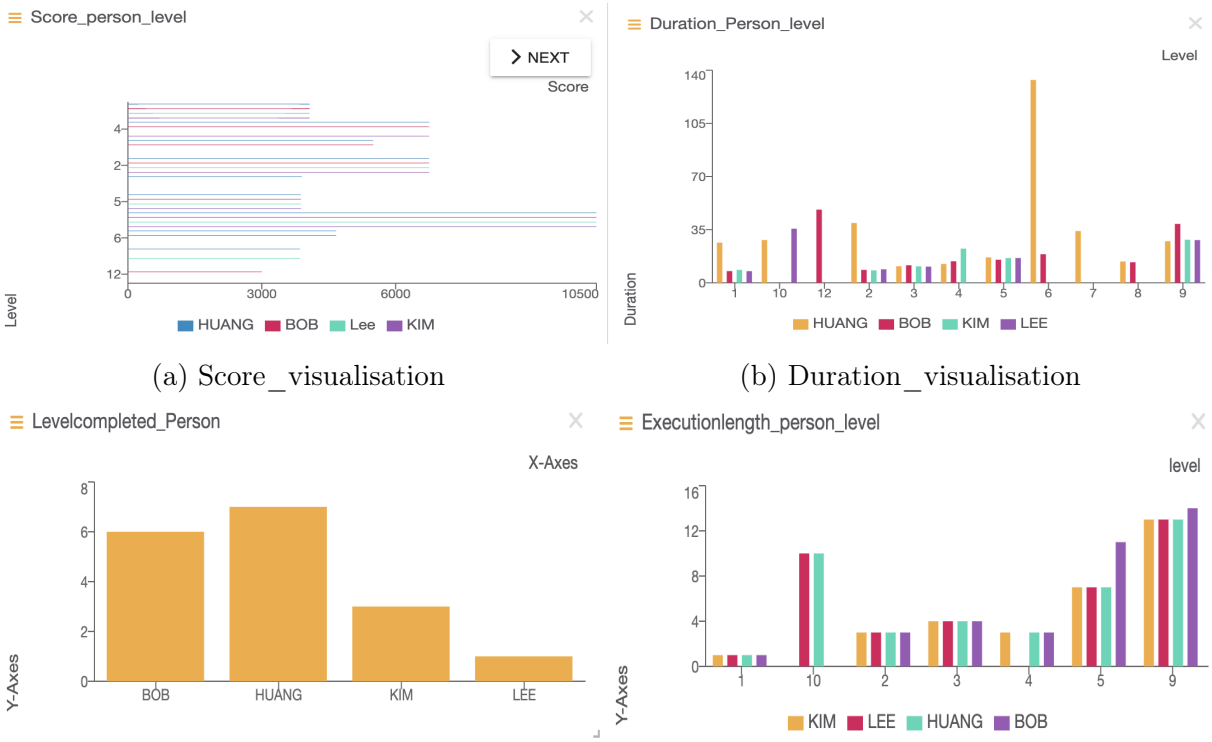

```
},
"extensions": {
  "https://w3id.org/xapi/gblxapi/extensions/invalid/level": [
    "3"
  ],
  "https://w3id.org/xapi/gblxapi/extensions/invalid/score": [
    "6000"
  ],
  "https://w3id.org/xapi/gblxapi/extensions/duration": [
    "8.78048"
  ],
  "https://w3id.org/xapi/gblxapi/extensions/nb_lv_completed": [
    "3"
  ],
  "https://w3id.org/xapi/gblxapi/extensions/code_length": [
    "4"
  ],
  "https://w3id.org/xapi/gblxapi/extensions/execution_length": [
    "4"
  ]
```


Figure 15: The part "extension" in a statement

We can read the figure below, which is the extension part of a statement sent by "Winlevel", we can learn from the figure that the player won the level 3 with a score of 6000 and the length of the code is 4. This level costs the player only 8 seconds and the total number of level completed by the player is 3. This information will help us draw figures for visualizations.

7.3 Visualizations

The visualization part is the most important part for the trace because the professors won't go to read statements one by one to figure out students'advancement. So in this game, we want that the professors can know about student's advancement by providing students' score, time cost, level_completed and execution length.



(a) Score_visualisation

(b) Duration_visualisation

(c) Levels_completed_visualisation

(d) Execution_length_visualisation

Figure 16: Exemple of Dashboard for professors to visualize students'avancement.

8 Conclusion

While we weren't able to finish everything we wanted in time. We've explored some new aspects that are surely not considered by everyone. We hope this project would inspire further experimentation, whether from the professors or future students.