UC Irvine

UC Irvine Electronic Theses and Dissertations

Title

iXercisePlus: Examining the role of Intelligent Agent and Multiplayer in exergaming

Permalink

https://escholarship.org/uc/item/2dw6q4vd

Author

Ji, Chenxing

Publication Date

2020

Copyright Information

This work is made available under the terms of a Creative Commons Attribution License, availalbe at https://creativecommons.org/licenses/by/4.0/

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA,
IRVINE

iXercisePlus: Examining the role of Intelligent Agent and Multiplayer in exergaming

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Chenxing Ji

Thesis Committee:
Professor Dr. Magda El Zarki, Chair
Professor Dr. Athina Markopoulou
Professor Dr. Fadi Kurdahi

2020

© 2020 Chenxing Ji

DEDICATION

To god and those who have shown their love and support throughout my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Throughout the way, many has contributed to this thesis and many has shown support and I would like to thank all those who have helped, encouraged and supported me in finishing this thesis.

I want to give my special thanks to my thesis advisor and committee chair, Professor Magda El Zarki for the awesome support she has given to me through out the way of completing this thesis as well as this wonderful and exciting opportunity to work and expand the existing iXercise platform. In addition to work, I also want to thank her for showing me tremendous support for making potentially life-changing decisions along with advice for life.

I would also like to give my thanks for my thesis committee members, Professor Athina Markopoulou and Professor Fadi Kurdahi for willing to spend their valuable time and giving me this opportunity to be my complete my thesis committee.

In addition, I would also like to thank those who have contributed to this thesis project that are also from UCI, Lydia Qiao and Cole Thomas Panning who contributed a lot to the idea of the intelligent agent system. I'm also thankful for the high-school students in the Vel-Tech program from Valencia High School, Donovan Nguyen, Samuel Chen, Eric Lee, Daniel Lee, Joshua Kim and Troy Quack who are brilliant and have contributed greatly on finishing the Overcook and Miner Quest.

Aside from work, I would also like to thank all those from the "Breakfast Club" whom I met and has been great friends with throughout the two-years of study here at UCI. Life of completing this master degree would be so much harder without you guys.

Furthermore, I would like to thank my parents for their support in the road of pursuing higher education and doing research.

At last, I want to give my very special thanks to Ms. Wu, Fei who has greatly helped me along the away and showing her tremendous support on finishing up this project and thesis.

ABSTRACT OF THE THESIS

iXercisePlus: Examining the role of Intelligent Agent and Multiplayer in exergaming

By

Chenxing Ji

Master of Science in Computer Engineering

University of California, Irvine, 2020

Professor Dr. Magda El Zarki, Chair

The iXercise exergaming platform was developed with sensors, micro-controllers, an exercise bike in association with the popular Minecraft game to promote physical activities among children. While iXercise platform has been developed and demonstrated a promising benefits towards promoting physical activities in children, the existing iXercise platform yet lacks contents that can attract children's interests and excitements. In this project, we further developed the existing iXercise platform with more multiplayer content, exciting quests and an intelligent agent system that can adjust the difficulty levels based on children's heart rate.

Chapter 1

Introduction

1.1 Background

Exergaming, a term that is used for prompting physical activities through gaming, has been widely researched in recent years.[6] This project is developed based upon the iXercise platform previously developed by Yunho et al. The iXercise exergaming platform aims to promote physical activities combing regular exercise with gaming, especially towards younger populations.[4]

1.2 Motivation

The original iXercise exergaming platform relied on a single player chasing task sequence to provide the sense of achievement for completing the quest. However, repetitive and tedious quest play would greatly reduce the amount of interest and lose attention of the player quickly. In addition, only providing single player quests limits the possibility of the player interacting with other players conducting the exercise session at the same time.

Thus, our approach to extend the current exergaming platform aims to add the multiplayer functionality to the quest system as well as adding additional interesting playable quest that is essential to the iXercise exergaming platform.

The original development of the iXercise platform also only set up the same difficulty level for players with different physical conditions, which result in some of the players with better physical conditions would be able to finish their quests than those with worse physical conditions. Therefore, in addition to the multiplayer extension and the playable quests, this project also aims to add an Intelligent Agent that can modify the difficulty of the playable quest automatically based on the player heart rate.

1.3 iXercise Overall System Description

This section describes how the overall system works, in particular, how each module is separated and how different modules interact with each other.

The overall system consists of the following four different modules, described from the exercise bike side to the Minecraft game side:

Raspberry-pi Middleware Java Application The Gateway Interface Java application handles the communication between hardware modules and the software modules. The application runs on a raspberry-pi physically connected to the exercise bike.

Bike Mod The Minecraft Forge mod handles the communication between the Minebike Mod and the Middleware.

MineBike Game Mod The overall Minecraft Forge mod that provides the gameplay and quests in this project.

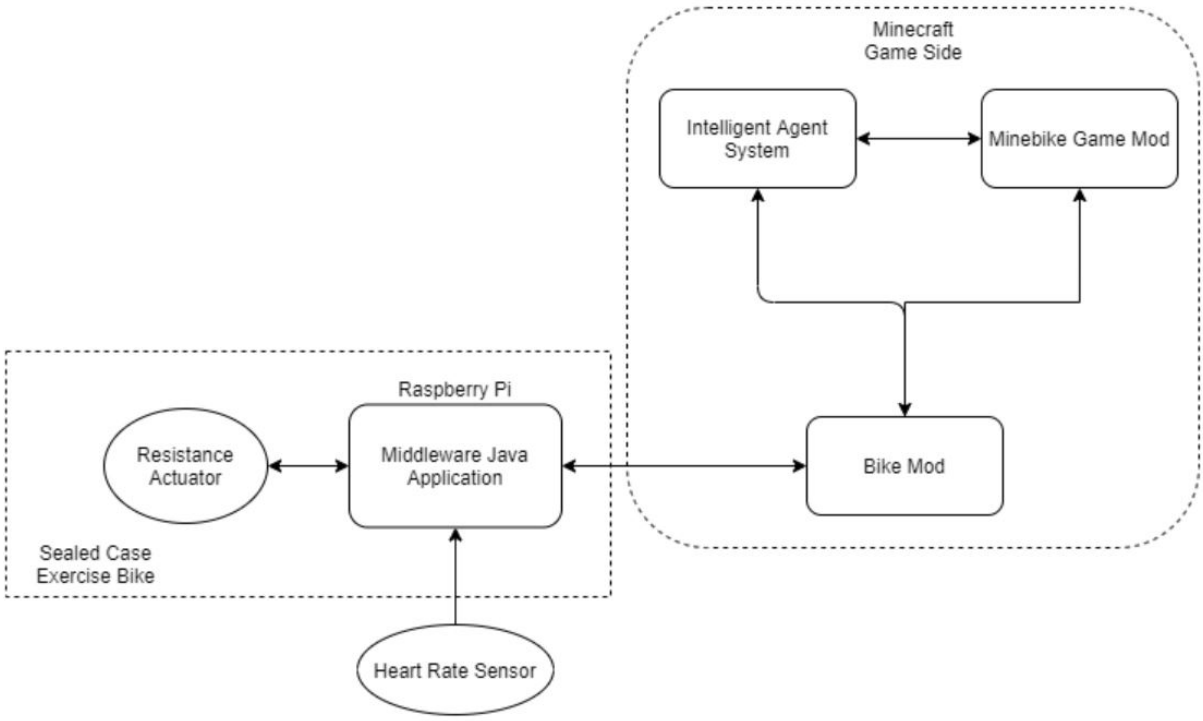

Figure 1.1: Overall System Overview

Intelligent Agent System The Intelligent Agent system is set-up to utilize collected data from both sensors and gameplay to provide tailored exergaming experience .

1.3.1 Raspberry-pi and Middleware Java Application

iXercise exergaming platform uses raspberry-pi to be the central controller device for the hardware side side of the system, both sensors and motor-tracking devices are connected to the raspberry-pi. Thus the raspberry-pi device is essential to the hardware side and its main functionality is provided by the middleware java application.

Physically runs on the raspberry-pi, the general functionality of the middleware java application is to performed as a centralizing communication hub that parses packets from different modules and passes on the information based on the packet specification. In addition, this java application also provides a general user interface for managing the application data.

While most of this application is done by YunHo in the iXercise project, our main new contribution is adding functionality for this application to transmit heart rate related data to the Minecraft Intelligent Agent.[4]

1.3.2 Bike Mod

The Bike Mod, on the other hand, is a stand-alone Minecraft Forge Mod that is separated from the main Minebike Game Mod such that it would provide function interfaces for the game mod to query exercises bike related information and the readability of the code base would increase. The Bike Mod also maintains a custom communication channel between the Middleware - Raspberry Pi and game for the AI system.

1.3.3 Minebike Game Mod

The Minebike Game Mod is the overall gameplay system that consists of several sub-modules. In the multiplayer system, this game mod is being ran on both the logical client side and the logical server side. Therefore the code required to be more carefully written. A more detailed explanation on the concept of sides can be found in Section 2.1. This mod contains the following: a Mini-Quests Module that handles the gameplay of each individual quest, a Intelligent Agent Module that would automatically adjust the game difficulty level based on the specified target heart rate.

1.4 Intelligent Agent System

The intelligent agent system is capable of making decisions on whether the current on-going game difficulty level needs to be adjusted based on the input of the heart rate sensor and the

resistance sensor. In addition, this intelligent agent system also have a general user interface components attached to display related information and the player might feel interested in learning.

Chapter 2

Systems of Minecraft Forge Mod

This chapter mainly describe in detail how the software architecture works under the general system design as well as provided explaining the code base for all the different levels of code modules in the project.

The mod consists of a main class for the mod that handles registering the mod so that the forge mod loader would recognize the mod when the game started. A NPC module that handles the custom NPC registering and different. An AI module that handles the automatic adjustment of quest difficulty based on heart rate. Even though the AI module code is implemented within the Forge Mod, the AI system is quite complicated this chapter would not explain AI module and a detailed description of this is provided in Chapter 4.

2.1 Key Concepts of Minecraft

To explain how the multiplayer version of the exergaming platform is implemented, some of the key concepts is necessary for understanding. This section is dedicated for explaining these key concepts in Minecraft that are critical for understanding the source code and the

following sections in this chapter. These concepts are also the basic yet important concepts to keep in mind when programming any Minecraft Forge Mod.

In addition to the following explanations, the Forge Documentation also provides much relevant information on their website for Minecraft modders to reference.[2]

2.1.1 Sides

For running a Minecraft multiplayer mod, it is essential to keep in mind that the Minecraft game runs on two different instances: Server and Client.[3] Even though the same source mod code is present on both side, the not all the Minecraft resources are present on both sides. For example, the server is used for handling entity update while the client is used for handling GUI modules. Therefore, calling any GUI related methods on the server side would immediately crash the server.

The *isRemote* variable is used for tracking the logical side the mod is loaded, and before executing any side specific functions this variable should be checked to make sure the functions are being called on the correct side.

2.2 Main mod Class

The main mod class consists of several functions extended from the Minecraft Forge Mod interface functions to initialize data the mod requires. The functions subscribed to standard Forge Mod Loader events that trigger when loading the mod. A more detailed description is provided in the following:

preInit This function subscribes to the $FMLPreInitalizationEvent$ and handles the

pre-initialization phase such as allocating a the common proxy object and registering the event handlers for the mod.

init This functions subscribes to the $FMLInitializationEvent$ and instantiate the HUD-Manager only for the client side of the mod.

postInit This functions subscribe to the $FMLPostInitilizationEvent$ that instantiate and initialize all the manager classes.

There are a couple Minecraft Mods used by the project to enhance the gameplay of the project quests. The Soccer Mod used written by MrCrayFish by the soccer quest is used to reduce the work of implement our own soccer ball.[5] The fishing quest also used Fishing-MadeBetter Mod made by TheAwesomeGem for more aesthetically pleasing user interface.[8]

2.3 Events

The common event handlers are mostly managed inside the $CommonEventHandler$ class that are subscribed to events registered on the Minecraft Forge event bus. This section would mainly cover the events handlers subscribed to the Forge Event Bus as well as the functionalities of each event in our mod.

2.3.1 onEntityInteract

This event is triggered when an entity inside the game is interacted by the player and used for tracking which NPC entity is interacted by the player for triggering the correct NPC $onInteraction$ function to perform the correct action. Each of the NPC is mapped in a hash-map and by their unique names.

2.3.2 ServerTickEvent

The *ServerTickEvent* is only triggered on the server side. Currently this event is used for spawning the custom NPCs for this mod. In order to prevent the NPCs being spawned multiple times, the source code iterate over all the loaded entities inside the overall world and check if any of the entity's name matches with the NPC. Therefore, all the NPCs are required to hold their unique names to ensure the proper spawning logic.

2.4 Module Mangers

This section provides description for the manager class of each sub-modules inside the Mod. The module manager layer acts as an interface layer to provide a centralized usage for the resources and data within the module for others to use. The Module Managers includes a custom quest manager, a HUD manager and a NPC manager.

2.4.1 Custom Quest Manager

The Custom Quest Manager is the centralized manager class for all the mini-quests. The manager class utilizes the base interface class that all the quests should extend from to achieve the goal of providing simplicity of the code as well as providing readability of the source code. The quest manager class contains a hash-map that maps each quest object with its associated dimension id to provide access to each of the quest object to all the code not inside the scope of the quest module. The hash-map also provides $O(1)$ access to the quest object based on the property of the hash-map.

One additional benefit of having a centralized manager is that by utilizing the quest map, each of the quest no longer needs to individually subscribe to the event handlers. This can

be achieved through using the quest hash-map and wouldn't cause huge overhead as well due to the $O(1)$ access time of hash map. The remaining of this section would describe each of the function and reason about why this particular event is necessary for the quest design.

findAndStart

$findAndStart$ function is triggered on the logical client side when the client received a packet from the logical server side with a Enum value of $QuestStart$. When the this function is executed, the manager would use the quest hash-map container to execute the correct function on the client side.

OnWorldTick

The $OnWorldTick$ subscribes to the $WorldTickEvent$ on the Forge Event Bus. Every world tick function call would then be mapped to its corresponding quest world tick function inherited from the base class. The function executes every game tick and is used for tracking the server side data that needs to updated frequently.

OnPlayerTick

On the logical client side, instead of the $WorldTickEvent$, $PlayerTickEvent$ triggers on the Forge Event Bus and thus $OnPlayerTick$ function is mapped and used for client side data that needs to be updated frequently.

2.4.2 HUD Manager

HUD Manager class is responsible for managing all the GUI elements on the client side. The HUD Manager class both provides interface function for GUI elements to be registered or unregistered and handles the update of each shape within the container based on the Forge *RenderGameOverlayEvent*.

The HUD module provides three basic shape elements, rectangle, string and textures. The rectangle can be used for drawing shaped bars and such, while the string can be used for displaying text. As for HUD texture, the class is used for displaying

The HUD manager is also responsible for preventing the concurrent modification for the HUD shapes container such that the *OnGameOverlayChange* function would not cause java concurrent modification error. To achieve such goal, the java synchronized keyword is used for allowing single thread execution for game overlay change.

HudShape

HudShape is the interface class that extends from the Minecraft GuiScreen class. As the interface base class, the HudShape provides methods for all the extended classes to de-register themselves from the *HudManager* shapes container such that they would no longer be drawn on the screen.

HudRectangle, HudString and HudTexture

Extended from the HudShape class, all three classes are required to implement the *draw*() function to draw the corresponding shape on the screen. In addition, for managing the shapes easier, the constructor of all three extended classes register themselves inside the

HUD manager for code simplicity. To remove the current shape from the GUI screen, one would simply call the *unregister* method of the current class.

2.4.3 NPC Manager/Database

For the NPC sub-module inside the mod, the manager class acts more like a database for the NPCs to be registered and therefore the NPC modules utilizes a NPC database to keep track of the NPCs inside the overall world. The actual implementation of the NPC database utilizes the default built-in hash-map in Java that key is the name of the NPC and the value is the actual object of NPC class itself.

NPC database for Mini-Quests	
NPC Name	Corresponding Quest Name
Jaya	Soccer Quest
Ada	Fishing Quest
ChefGusteau	Overcook Quest
Renzler	Tron Quest
Elon	Miner Quest

Table 2.1: NPC Database with

NPCEventHandler

The NPC event handler deals with the *EntityInteract* event such that the target entity is registered inside the database, and would subsequently triggers the corresponding NPC's *onInteraction* function to perform its functionality.

NPCUtils

This is the Utilities class for NPCs that provides common functionality for all the NPCs. Currently, the provided utilities include a spawn NPC function for spawning NPC with

specified position, name and texture in the specified world.

2.5 Communication Specification

This section is dedicated for explaining the communication specification between the two sides of Minecraft, Client and Server. As explained before, since the logical server side and logical client side can be run on two different physical machine, the communication channel is then established through a network channel and each communication needs to be transmitted through a internet packet. The communication specification also consists of two parts, specifies the meaning of the ENUM value at the beginning of each packet.

2.5.1 Server ENUM Specification

Server specification is used for the packets transmitted from the logical server to the logical client, and the class is called *EnumPacketServer* inside the *constant* folder. Figure 2.2 describes the explanation of each value.

ENUM value Server	
Enum Name	Explanation
SoccerQueueingTime	Remaining wait time on client
SoccerLeftScoreUpdate	Blue team scored
SoccerRightScoreUpdate	Red team scored
QuestStart	Quest starts, followed by Dimension ID
QuestEnd	Quest ends, followed by Dimension ID
QuestJoinFailed	Quest waiting/started, followed by Dimension ID
FishRetract	Fish hook retract button clicked
FishDistance	The current distance of a caught fish

Table 2.2: Server Packet ENUM specification

2.5.2 Client ENUM specification

Client specification is used for packets transmitted from logical client to the logical server with the class name *EnumPacketClient* inside the *constant* folder. Figure 2.3 provides the Enum names on the client side and the associated function name and its usage.

ENUM value Client	
Enum Name	Explanation
PlayerJoin	The player interacted with NPC to join quest, followed by Dimension ID
Fishing Distance	Initial distance of the fish

Table 2.3: Client Packet ENUM specification

2.6 Mini-Quests Design and Dimensions

This section is dedicated for a description on how the mini-quests design choice is made and an explanation on the quest related source code. Note that all the quest classes inherited from the same base class *AbstractCustomQuest*, which specifies a series of abstract interface functions and common shared member that the child classes can utilize. Since the mini-quests are now implemented as multiplayer mini-game, many code choices needs to be altered from the original Minebike-Mod to adjust to the client and server model.

To make each of the quests being able to reset to its initial state for each time the quest is played, each quest is then implemented on a different dimension such that quests wouldn't interfere each other.

2.6.1 Soccer

The Soccer quest is divided into three different phases, the initial state, the waiting state, and the gaming state. The three states forms a cycle such that initial state can only go to waiting state and waiting state can only go to gaming state, while the only state right after gaming state is the initial state.

The transitional flag is game state is controlled by two boolean variables $isWaiting$ and $isStarted$, corresponding to waiting state and gaming state.

Initial State

First state is the initial state where no player has joined the quest. When the first player tried to join the game after interacting with the NPC Jaya, the game state would transition from the initial state to the waiting state inside the $onPlayerJoin$ function.

Before waiting state actually started, the server side would teleport the player to the designated location inside the soccer field. Then the code would also apply two potion effect to the player to prevent the player from moving and jumping for the entire waiting time. Server side would also calculate the remaining wait time and send it to the client side as a network packet for displaying the remaining waiting time. On the client side, after receiving the remaining seconds to wait, it would create a count-down clock on the top of the screen and start the waiting state.

Note that each player might join the quest on different time over the waiting period and cause the count down to start at a different time. Therefore, correctly sending the remaining waiting time is essential for correctly syncing all the players to finish the waiting state around same time to ensure the fairness of the mini-game.

Waiting State

After entering the waiting state, the quest would utilize the both $onServerTick$ and $onPlayerTick$ function to start the count down towards the gaming state as the following figure shows. The waiting state last for thirty seconds that allows other potential players to join the same game session for multiplayer purpose. The server side code would handle the synchronization of the count-down time of each player for fairness.

Figure 2.1: Soccer Waiting State User Interface

Gaming State

The gaming state of the soccer quest lasts for five minutes where both teams are able to score as much as they can within the time period. The trigger for scoring is handled mostly on the server side after the ball remained inside the goal area for about half a second. Once the scoring method triggers, the server would re-spawn the ball into the center. Then send packets to each player updating the scores on the HUD.

At the end of the gaming state, the $end()$ function would be triggered to teleport all the players to the location where they were before joining the quest and also re-initialize the soccer world for the quest to be run next time.

Figure 2.2: Soccer Gaming State User Interface

2.6.2 Fishing

The Fishing quest generally requires the player to fish a number of different types of fishes over a limited time period.

Playing the Quest

To start the Fishing quest, the player would need to talk to a NPC called 'Ada' inside the overall world. Once the NPC is interacted, the player would then be teleported to the target dimension based on the chosen difficulty level. Upon teleportation, the quest would also spawn a custom fishing rod on player's main hand. The intelligent agent system then would utilize the heart rate data and this fishing rod to spawn the correct type of fish for the exercise.

Fishing and HUD

During the fishing phase, the regular player movement would instead be replaced by a filling power bar such that the power bar represents the intensity of the fishing rod being rolled back. The player would need to maintain the power bar above certain level to catch the

fish. Since each fish has its own associated distance and catching the fish require zeroing the distance with the fish, the player would have to maintain above the power level for the same amount of time.

2.6.3 Overcook

The idea of the Overcook quest originated from the Overcooked Game that is playable on multiple gaming platforms. [7] The quest is divided into two simple states, a waiting state and a gaming state.

Waiting State

During the waiting state, the player would be teleported to a platform waiting for the game to start with a count down of fifteen seconds.

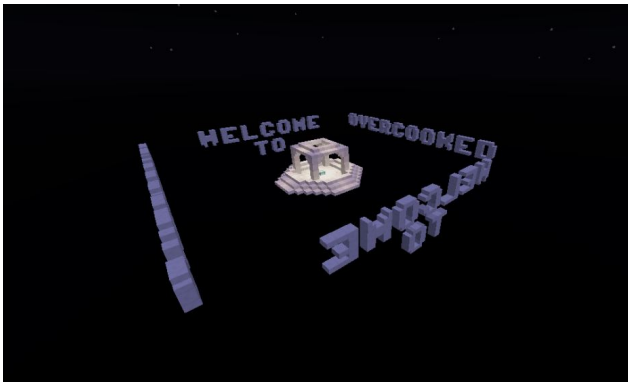

Figure 2.3: Overcook Waiting Room

Gaming State

During the gaming state of the overcook quest, the player would be prompted with a serious of recipes to collect ingredients and cook. Each recipe would require different ingredients and are set with a limited time to finish cooking the recipe.

There are different stations inside the game, and ingredients need to be collected from different stations to be able to cook the food. Once all the ingredients are collected, the player needs to go to the assembly station for cooking the ingredients into a food item. Figure 2.4 shows an overview of the gaming dimension.

Figure 2.4: Overcook Waiting Room

After successfully cook the item on the recipe, the player can go to the restaurant and interact with the manager for delivering the food to the customer.

The stations inside the game are intentionally set apart such that the player would be able to get enough exercise going from stations to stations.

2.6.4 Miner

The Miner Quest consists of five different lanes of different difficulty levels ranging from very easy to very hard. The objective of the Miner quest is to complete the lane within a certain

time period while there are lava chasing behind you.

Each lane has its unique design such that each lane have a different difficulty level and environment for players to have varied exergaming experience. Below includes a sneak-peak of what each lane looks like ranging from easiest to hardest.

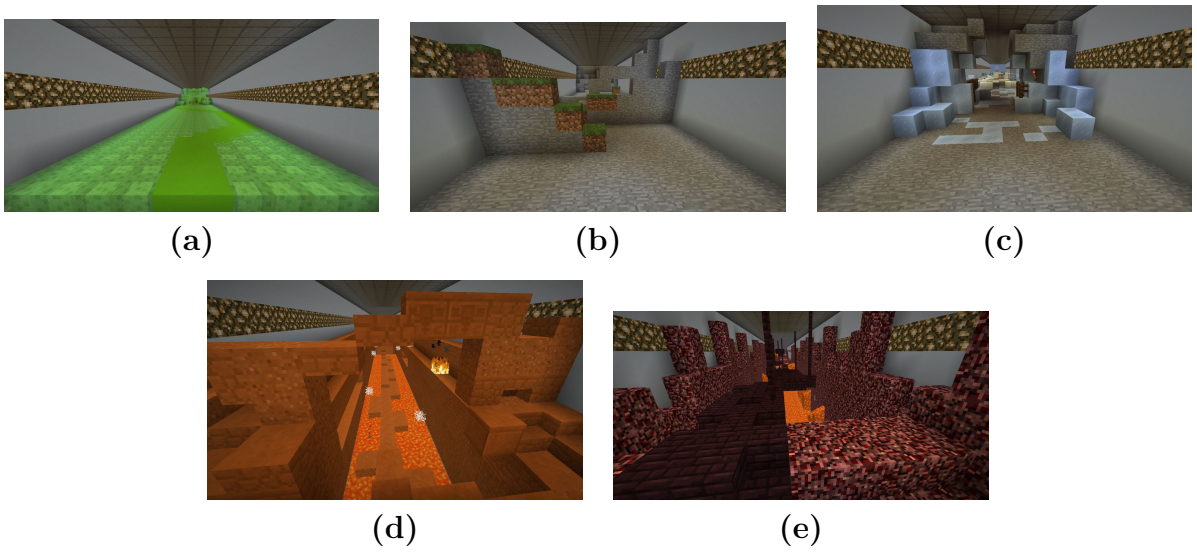

Figure 2.5: **(a)** Lane 1 **(b)** Lane 2 **(c)** Lane 3 **(d)** Lane 4 **(e)** Lane 5

2.6.5 Tron

The objective of the Tron Quest is to compete with a NPC player called "Rinzler" and try to corner him with the glass pane that is generated along your moving path. Meanwhile, the NPC would also put down glass pane right after him and try to corner the player. The objective is corner the NPC to win the game. Since the NPC moves relatively fast, this quest requires quite a lot of physical exercise with the bike to win. This is where the AI module comes into play and a detailed explanation is provided inside the Intelligent Agent System section.

Game Mechanism

The quest utilizes *onPlayerTick* event to detect player's location every tick, and put down glass pane automatically behind both the player and the NPC "Rinzler". The quest also constantly checks whether player or "Rinzler" remains at the same location for over five seconds to determine the winner of the game.

Chapter 3

Middleware and Bike Mod

This chapter is dedicated for explaining the role of the middleware system and the bike mod. The middleware system runs physically on a raspberry-pi embedded system and the Bike Mod runs with the Minecraft game on the client side.

The middleware system in the iXercise platform serves as a communication hub that handles the flow of communication and the physical device that the middleware deploys on is a Raspberry Pi embedded system that utilize many functionalities provided by the Linux operating system. The system mainly consists of two different parts, a gateway Middleware Java application that handles the communication, and a serious of bash scripts that utilize the operating system.

The bike mod is the mod on the game side that maintains the communication channel with the middleware along with the modifications that enables the player character to move using the bike. The bike mod is a simplified version of Yunho's Minebike mod where only the code related to the bike is kept.

3.1 MiddleWare Java Application

The middleware java application mainly still uses the gateway application developed by Yunho in the iXercise gaming platform. There are only a few modifications applied in our extension to this java application. For example, while testing the Bluetooth module, since gaining physical access to the bike can be hard under the current COVID-19 pandemic, we modify the application such that the heart rate data can be transmitted to the game without the bike.

3.1.1 Middleware Pi Scripts

The section describes the what each bash scripts on the middleware device does and how they relate to each other for the middleware to function.

- ChooseBT The file reads from the file *BT.dat* and choose the corresponding MAC address for connecting the heart rate Bluetooth monitor.
- ReadHeartRate Tries to connect to the chosen heart rate monitor based on the MAC address. If successful and the heart rate is not zero, it would then use the Linux command client to transmit the heart rate data to the Middleware Java Application.

3.2 The Bike Mod

The Bike Mod is a separated Minecraft Forge Mod away from the main gameplay Minebike Mod. The Bike Mod would be imported as a library for providing functionality to establish the communication channel with the middleware.

To make the Bike Mod light weighted and increase the readability of the code base, the

Bike Mod is derived from the original Minebike made by Yunho such that most of the codes related to quests and gameplay are removed.

Using a software layered design, the Bike Mod act as a lower level interface system for the gameplay Mod and the Intelligent Agent system to query bike data such that the Minebike Mod would not be able to directly access bike related data. In the case, the resource regarding the exercise bike on the Minecraft side can be well managed such that data race and concurrent modification to any shared data would not occur.

Chapter 4

Intelligent Agent (AI) system

The AI module of the Minebike system, with design of a central Intelligent Agent that not only provides communication channel for the different sub-modules of the AI system, but also servers as a central data hub and an layer of abstraction for accessing sensor data. The central agent is the key element to the design of the AI system such that the system complexity is reduced and is modularized with this centralized design.

The overall AI system consists of five different parts: sensory inputs, diagnostic system, central agent, user interface, and game controls. Figure 4.1 below illustrates how the different parts of the system interact with each other, and the arrow indicates the data flow between each objects.

4.1 Sensory Inputs

The sensory incorporate two different sensors, a heart rate sensor and a resistance actuator. The heart rate sensor is connected through Bluetooth with the Middleware, while the resistance actuator is connected through one of the GPIO pins with the Middleware. Since

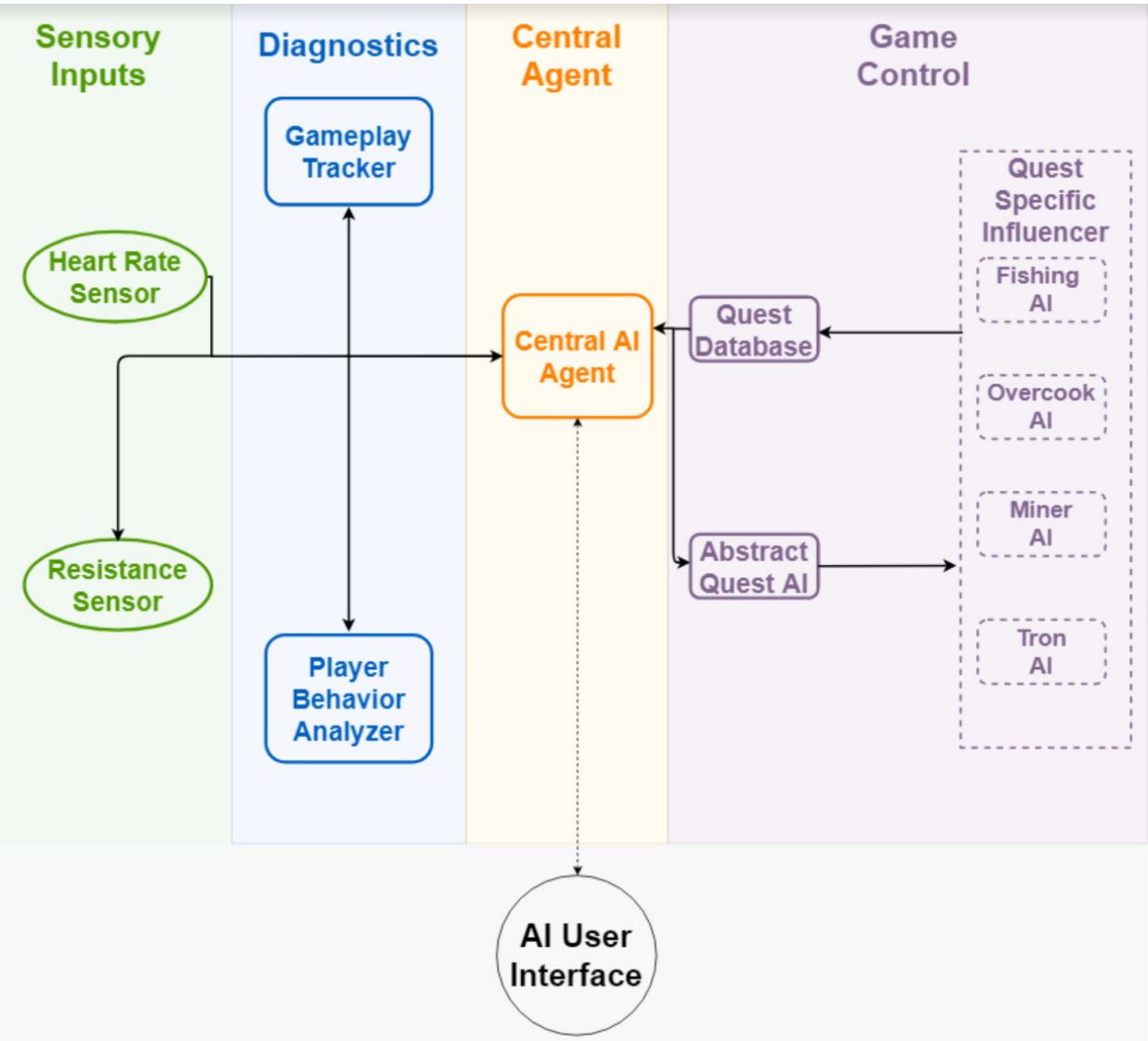

Figure 4.1: AI system overview

the Middleware maintains a communication with the Client Side of the game, the sensor data would be passed on to the game for central agent to encapsulate and provide a layer of abstraction towards upper layer usage.

4.1.1 Heart Rate Sensor

The heart rate sensor is of Model H9 made by Polar. While the middleware system utilizes the built-in support of Bluetooth Low Energy(BLE) module inside Raspberry Pi to establish the connection with heart rate sensor. [1] Once the sensor detected the heart rate pluses, it would transmit the current heart rate to the middleware.

4.1.2 Resistance Actuator

The resistance actuator sends the data through one of the GPIO pins on the middleware, and gets transmitted to the central agent. However, instead of the single way communication like the heart rate sensor, the resistance sensor maintains a two-way communication, where the central agent can also influence the resistance sensor. Thus, the central agent is capable of modifying the bike resistance dynamically for physical activities.

4.2 Diagnostics

The diagnostic system consists of a gameplay tracker that stores any previous gameplay related information and a player behavior analyzer that suggest games based on the gameplay history recorded by the gameplay tracker.

Unlike other physically visible sub-modules, the diagnostics system acts as a hidden layer in the AI system where the system does not output a visible outcome. Inside, both diagnostic subsystem periodically issue direct query to gain information from the central agent and feed back the diagnostics back to the central agent.

4.2.1 Gameplay Tracker

The gameplay tracker saves the previous played quests for each player over multiple pediatric sessions. Since the data stored in RAM on the client side would be lost on player log out of the game and one player may login to a different machine over different sessions, the stored gameplay tracker data would need to be transmitted between client and server on two occasions, when player join game and when player quit game.

4.2.2 Player Behavior Analyzer

The player behavior analyzer queries the central agent for the related gameplay data and the registered AIs for analyzing the play history of this user to suggest a quest for the player to exercise.

4.3 Central Agent

This section illustrates the Central Agent class for the Intelligent Agent. This agent maintains its access to input sensors, diagnostic modules and quest control and servers as the central communication hub.

The central agent periodically queries the sensor inputs for the current heart rate data and the current resistance value. Then the agent stores a copy of the data until the next query happens for the rest of the modules to use. This period happens every one second and since the heart rate sensor only transmit heart rate data every second and the bike resistance actuator takes time to take effect, this period setting is sufficient to maintain updated data.

In addition, the central agent also manages a trigger for the Player Behavior Analyzer mod-

ule. If the player spend five minutes not participating any of the quests while not being able to maintain his/her target heart rate set by the pediatrician. The central agent would trigger the player behavior analyzer to get a suggested quest for the player. Note that if the player chose not to play this specific quest, we are not able to force the player into physical exercise. Instead, we can only predict what quest the player likes and encourage them to participate and exercise.

4.4 AI system User Interface

The AI system user interface have the following four components

- Heart rate Display: Display the current heart rate
- Target Heart rate: Display the target heart rate specified by pediatrician
- Target Reached Indicator: Indicates whether the target heart rate has reached the predefined amount of time for the current session
- Resistance Indicator: The resistance indicator displays the current resistance value read from the resistance actuator on the bike.

The user interface of the AI system is being displayed when the player stays inside the overall world and none of the Mini-quest is started. As the quest starts, each quest specifies its own user interface label which would make the user interface look crowded if the general AI system user interface is being displayed as well.

The figure below demonstrates what the AI user interface looks like while playing the game:

Figure 4.2: Intelligent System HUD

4.5 Game Control

Game control system is made up of two central module, a quest AI database and a abstract quest AI class, and four quest specific modules, Fishing AI, Overcook AI, Miner AI and Tron AI. The quest AI database maintains a list of all the registered quests that has an implemented AI module attach to the quest. Abstract Quest AI class is another essential piece that provides some common functionality as well as specifies a serious of functions that each extended class would be required to implement for the central agent to receive necessary information.

The sections below briefly describes what each individual AI module implemented for each quest does as well as how it influence the gameplay based on the heart rate.

4.5.1 Fishing AI

The fishing AI implemented inside the fishing quest is for adjusting the spawning rate of each fish based on the heart rate of the player. Since each fish has its own difficulty level to catch, If the player heart rate was not able to reach the target goal on the selected difficulty level, the AI module would influence the game to have a higher spawning rate for bigger fishes that the player needs to pedal harder to catch. Vice-versa for the player heart rate being higher than the target rate for reducing the possibility of the player over-exercise.

4.5.2 Overcook AI

The overcook AI is designed for dynamically modifying the time constraint for each recipe while playing the overcook quest.

4.5.3 Miner AI

Since the Miner quest uses a timer for the time taken to complete the lane, the AI module of this quest is implemented to affect the next run of the quest such that the next run of the timer would be based on the previous run in combination with the previously recorded heart rate.

4.5.4 Tron AI

The AI module of the Tron quest aims to dynamically modify the speed of the NPC "Rinzler" that compete with the player inside the quest. With the help of the AI module, the quest can tailor the difficulty level easier toward each individual player without necessity for manual difficulty adjustment.

Chapter 5

Summary and Conclusion

This chapter summarize the work of the thesis and provides a conclusion on the current status of the Minebike exergaming project. In addition, this chapter would also include a section talking about the potential future work from this current point of view and what impact and improvement these work might bring to the current system.

5.1 Future Work

With the current Intelligent Agent system storing the gameplay through the gameplay tracker, additional work on analyzing the player behavior could be done and added to the current project. For example, an achievement reward system can be implemented upon AI system to further promote physical activities to the players by giving out various in-game rewards. Note that this is just simply one potential idea about the expansion of the Intelligent Agent system and many other ideas can be implemented as well.

On the gameplay side, work can be done with the current quest system to expand the current quests system by adding more playable quests or potentially create an attractive story-line

for the player to immerse themselves more into Minebike games and physical activities.

5.2 Summary

In conclusion, this thesis presented the design of a multiplayer quest system for Minecraft as well as the outline for the Intelligent Agent system for prompting physical exercise through gaming.

In Chapter 2, a detailed walk through of the software structure for the in-game quest system is provided. In order to promote physical activities through quests, most quests are implemented with different difficulty levels for the Intelligent Agent System. Moreover, the soccer quests are implemented with multiplayer capability for the players to be competitive and thus increase their interest in physical exercise.

Chapter 3 describes the role of the middleware raspberry pi and bike mod inside the current system and how they are connected with the rest of the system.

In Chapter 4, the Intelligent Agent system is presented. The Intelligent Agent system read real-time data from different sensors and incorporate such data to influence the gameplay such that each player can have a automatic tuned exergaming environment that suits the player's physical condition.

Bibliography

[1] Rpi bluetooth le. https://elinux.org/RPi_Bluetooth_LE, Last accessed on 2020-09-07.

[2] Minecraft forge documentation, 2020. https://mcforge.readthedocs.io/en/latest/, Last accessed on 2020-09-07.

[3] Sides in minecraft, 09 2020. https://mcforge.readthedocs.io/en/latest/concepts/sides/, Last accessed on 2020-09-07.

[4] Y. Huh, G. T. Duarte, and M. El Zarki. Minebike: Exergaming with minecraft. In *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*, pages 1–6, 2018.

[5] MrCrayfish. Soccer mod 1.12.2 (playing football in minecraft), Sep 2019. http://www.9minecraft.net/soccer-mod/, Last accessed on 2020-09-06.

[6] Y. Oh and S. Yang. Defining exergames & exergaming. 01 2010.

[7] Team17. Overcooked. [Digital & CD-ROM], 2016.

[8] TheAwesomeGem. Fishing made better, Dec 2018. https://www.curseforge.com/minecraft/mc-mods/fishing-made-better.