AI-TEM: TESTING AI IN COMMERCIAL GAME WITH EMULATOR

Worapoj Thunputtarakul and Vishnu Kotrajaras
Department of Computer Engineering
Chulalongkorn University, Bangkok, Thailand
E-mail: worapoj.t@student.chula.ac.th, vishnu@cp.eng.chula.ac.th

KEYWORDS

Testbed , Artificial Intelligence, Commercial Game.

ABSTRACT

Many artificial intelligence (AI) game researchers find that it is difficult to find a game environment that they can appropriately test their AI on. They usually have to develop parts of an existing game, using tools that come with the game. Some even have to re-write their testing game from scratch. Finding a perfect game environment that one can use to test his AI is not easy, especially if a commercial-quality game is required. Huge amount of time and effort are lost in finding such ideal testbed. This paper presents AI-TEM environment, a testing environment for testing AI by using console game emulator and its ROM data to simulate and run a commercial game. AI-TEM can be used to plug many AI onto many commercial games. Researchers interested in higher-level abstractions of game AI can test their already developed AI algorithm on a commercial game. We believe that AI-TEM adds a wider range of possibilities to AI testing.

1 INTRODUCTION

Research and developments related to computer games have always focused on graphical technology. However, players have begun to demand for more playability. Recent games have incorporated smart AI into their gameplay and became very successful because of that. Therefore, research in AI is important for the game industry. On the other hand, games provide interesting testing environments for AI researchers.

One problem faced by many researchers is to find or develop a proper game environment to use in their AI testing (Graepel et al 2004, Kendall and Spoerer 2004, Ponsen et al 2005). A game that should be used to test AI should have the following qualities: It should be a game that has many ways to play, many ways to win, and the game should be complex enough to separate expert players from novice players. It should be a commercial quality game. Because if researcher's AI can win against its initial game AI, then researchers can claim that the newly developed AI truly has enough quality and efficiency to use in commercial game (Spronck et al 2004).

There are testbeds developed for testing AI (Aha and Molineaux 2004, Bailey and Katchabaw 2005), but many of them do not come with a complete game ready to be used. Researchers will have to find a game or game engine to integrate with it. Large amount of time may be used.

This paper proposes another way to test game AI in an environment that has been well made and well designed, called AI-TEM (AI-Testbed in EMulator). AI-TEM uses an emulator of Game Boy Advance (GBA), developed to test many AI methods.

In this paper, an emulator means a game console/handheld emulator that simulates the working of game console/handheld hardware such as GBA, PlayStation, and arcade machine on any personal computer. There are many emulators of console/handheld game hardware. VisualboyAdvance (VBA) (VisualboyAdvance 2005) and VisualboyAdvance Link (VBA Link) (VisualboyAdvance Link 2005) are GBA emulators. ePSXe is a PlayStation emulator. Even arcade machines have MAME as their emulators.

ROM (Read Only Memory) is the game data dumped from the original game cartridge or disc. Using a game ROM with its emulator, a game can be simulated and played on PC.

We used GBA emulator, VisualboyAdvance, for developing a prototype of AI-TEM. And we used Street Fighter Zero 3 (STZ3) game ROM as our test ROM, so that we could experiment and write additional tools for a real example. VBA is open-source, therefore we can modify its functions. There are many interesting games on GBA that can be used to test AI. VBA also has plenty of resources, technical documents and support tools provided for us. The VBA Link is an extended version of VBA. The VBA Link team modifies original source code to make linkage possible. In this paper we will call both VBA and VBA Link as VBA.

STZ3 is a fighting game. In this type of game, a player must select one character from many characters, and fight one by one with an opponent character. A player must decide what action he will perform in many different situations based on his character and opponent character's status. Therefore, we believe this is a good game for tuning our testbed and for AI research.

2 AI-TEM FRAMEWORK

The concept of AI-TEM is generally simple but there is some low-level work involved. Researcher's AI may need to know game state data, such as object position or character animation, but it cannot access the source code of the game. The AI can only access the source code of the emulator. So we must get the game state data from memory data the emulator is emulating. We can see only a binary (hexadecimal) value of game data that changes in every cycle of a game execution. We must therefore find out which address stores the value that we are interested in such as position, animation, etc. We will use values in those addresses as game state data for AI testing. When we know the game state, AI can be written to react in each situation, by sending a controller input, or forcing memory address value. Using this concept, we can use AI-TEM as an AI testing tool.

Finding each address that stores those game state data is difficult if done manually. Some values can be found easily, while some are rather hard to find. User should have some knowledge about programming in order to be able to identify address more successfully. Some examples of how to find the address of game data are demonstrated below.

Example 1: Finding address of character's health. Starting by identifying all the values used in the game. Then the game is played and the character's health is forced to decrease. The value that represents the character's health should in fact decrease too. All game values are then searched and compared with values before the health decreases. It is common to find many values decreasing. The process should be repeated, with different values of health, until one address is identified.

Example 2: Character's bullet position. The concept is the same as the character's health example. When a bullet moves forward, its position value should increase continuously. But the time period that the bullet is alive is very short. Therefore, repeating the experiment as many times as one wants becomes difficult. If users must press a command every time that they want to find a value, it will not be convenient. A tool that can arrange this situation is needed, such as movie recorder (section 3.0).

We had developed some tools to help finding the address more easily and will describe it in section 3.0 below. There are other techniques to find values that we will not discuss in this paper.

Therefore, in the beginning phase of using AI-TEM, the user must find the address of game state data that their AI module needs to know. AI-TEM is depicted in figure 1 and 2, and discussed in more detail in the sections below.

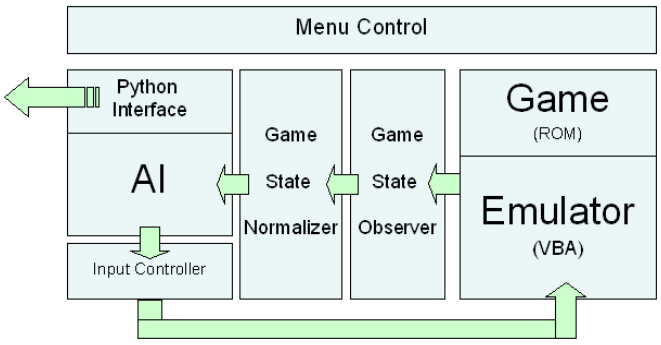

Figure 1: AI-TEM system overview.

2.1 Emulator Core (VBA)

The core of this testbed is VBA (Link) emulator. It is used to run game ROM and simulate the game. It also has many tools useful for getting game state and testing AI. These tools will be described in section 3.1.

2.2 Menu Control

We add a menu into the emulator to control the working of AI-TEM, to turn on/off AI module or switch between different AI modules. We can also activate other utilities that the system may want to use.

2.3 Game State Observer

A game state can be known by observing data on the memory address of the emulator and locating which address stores the data that we want to know. (In STZ3, game states that we are interested in consist of position of a character, position of the character's bullet, the character's health, and current animation of the character.) We implemented this module by modifying the memory viewer tools of VBA. Users can identify addresses and size of data (8, 16, 32 bits) that they are interested in. When AI-TEM is running, in every frame, Game State Observer will copy the values from those addresses to the data structure that an AI module can use.

2.4 Game State Normalizer

Before Game State Observer sends game state data to AI module's data structure, the game state data must be normalized or interpreted, depending on the game and format of data that we obtain from the memory. Example: For STZ3, we use address 0x20007C2 (16bits) as the address that stores the character 1P position in the X axis. The range of value that we got from that address is 44 (002C) to 620 (026C), 576 units. But when using it, we should normalize x position value to 0 - 576 for user friendliness. Therefore, we must subtract 44 from the value copied from the memory of the emulator. This normalization process is not necessary if researchers do not care about the format of raw data from the memory. We currently provide this module as a code template for users to modify.

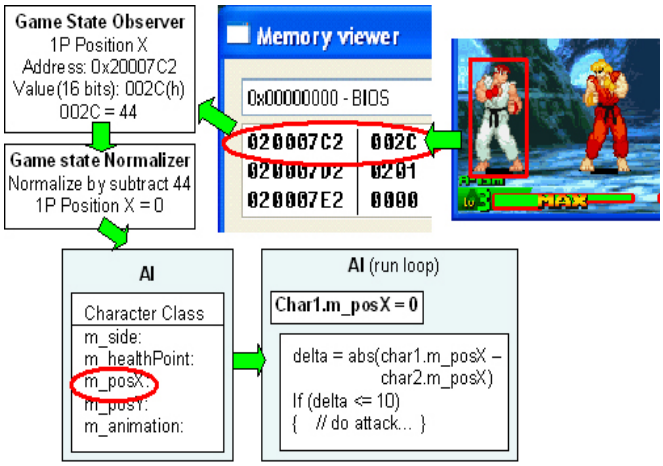

Figure 2: Work flow diagram of AI-TEM system.

2.5 AI

This module is where a user of AI-TEM will put his AI module in. In every cycle of emulation, the emulator will execute this module. This module evaluates the game data and decides what controller input it will send to the controller module. There is enough of VBA CPU power for calculating non-intensive work, such as script (Dynamic script (Spronck et al 2004), Genetic Programming result script, etc.). With extension, other AI methodologies can also be added. In our experiment with the system, we write two static scripts for testing the use of AI-TEM. The detail and result of this script will be shown below in section 4.1.

2.6 Python Script Interface

Python (Python 2006) is an interpreted, interactive, object-oriented programming language. It is also usable as an extension language for applications that need a programmable interface. Python is portable: it runs on many

OS such as Windows and Linux. It is one of the most famous script languages used in many applications.

We modified the emulator to have an ability to use python script language, providing interface functions for a script writer to obtain game state data and to control the game via any AI module. A script writer can write their python script separately without running the emulator, and can change script without recompiling AI-TEM. This will benefit users who want to test their AI with static script. If researchers can generate AI output in python script format, it can be tested conveniently without the need of rerunning the game. Example of an interface function used in the testing of STZ3 is shown below.

```
int GetCharacterPositionX (int C)
int PressButton(int button)
```

The first function will return a position in the x axis of character C. The second function will send a parameter 'button' to the Input Controller module. It allows a python script to command character. Below is the example of python script that uses those interface functions.

```
import myLib
def Main_AI_Run_Loop():
if (myLib.GetCharacterPositionX(P2) <10)
{
    return myLib.PressButton(PRESS_B)
}
return 0
```

`myLib` is a library of interface functions that we provide from AI-TEM, it allows user of python script to use function `GetCharacterPositionX` and `PressButton`. This script results in character kicking (press B) when its opponent comes closer than a specified threshold.

2.7 Input Controller

The original VBA captures signals from joystick or keyboard and send them as input to a game. We modified the system so that our AI module can replace input signals from normal controller with its own signals.

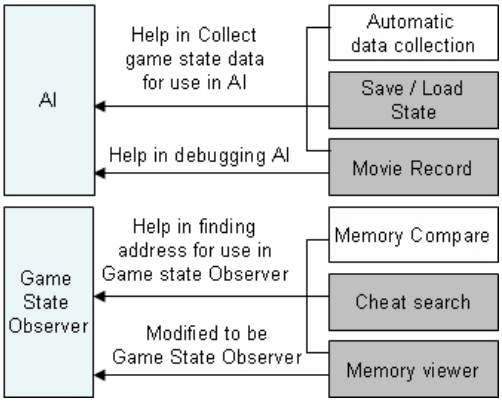

Figure 3: Usage of tools in AI-TEM system.

3 AI-TEM UTILITY TOOLS

Other than modules described in section 2.0, AI-TEM has other utility tools that can help in many tasks. Some of them are original VBA tools that we use in our testbed. Some are modified tools we made for our own use. The working of these tools is shown in figure 3 and described below.

3.1 VBA Tools

These are original VBA tools that we had used in AI-TEM.

Memory Viewer: used for displaying content of every memory data address in a variety of formats, 8, 16, 32 bits, sign, unsigned and hexadecimal. Our Game State Observer is modified from this tool.

Cheat Search: this tool is originally used for finding an address of data that we want to find, by searching all of memory and finding a value that matches a condition given by user. For example, users can use it to find a value in the address that is equal, greater or less than some specific value. In AI-TEM, this tool is used to help in finding address that Game State Observer will observe.

Movie Recorder: this tool can record game movie in two formats. VMV format will record only initial game state and inputs given by controller. It has a very small file but can playback only in emulator. AVI can playback in many movie player programs but its file is larger and uses a lot of CPU power. Movie recorder can help in a data collection process (section 3.2) and can help recording the testing output or debugging.

Save/Load State: When running a game in the emulator, the game state can be saved and reloaded to continue to play at the same point where it was saved. This ability is useful when researchers want to test decision conditions of their AI. They can save game state before their AI makes decision and can reload it to try another decision in perfectly the same situation.

3.2 Modified Tools

Memory compare tool: As said in section 2.0, finding address of value that AI module needs to know is difficult. Therefore, we modified the original VBA tools to be Memory compare tool. This tool will help in finding an address of data, by comparing many game states data, given a condition of data that users are seeking. Example: A user wants to find the address of character's health in STZ3. He will dump game states of various situations from the emulator. We define the game state of situation N as GSn.

```
GS1: character's health is 100%.
GS2: character's health is 50%.
GS3: character's health is 75%.
```

The user will set conditions of the value he wants to find. In this case, a health value address will have conditions as follows: The value in our required address from state GS1 must be greater than the value from state GS2. The value in our required address from state GS2 must be less than the value from state GS3. And the value in our required address from state GS3 must be less than the value from state GS1. This tool will compare every data in those game states and find the address that matches all user-given conditions automatically, without any need to run the original cheat search (The cheat search tool requires users to repeatedly experiment and find any address manually.). If users provide enough game states and proper conditions, finding a required address should be straightforward. We believe this tool can save a great deal of time finding those values.

Automatic Data Collector: There are some situations that we want to collect a lot of data from game state. It is difficult to collect them manually. Example: In order to imitate human reaction and decide its next response, an AI module must know the animation of both characters. For example, if an opponent character is going to punch, our character must detect the opponent's movement and perform a guard. After knowing the address that stores values of character animation, we need to find out which value in that address corresponds to which animation. (for example, 0 means stand, 72 means crouch, 124 means jumping) Therefore, we need some tools to help collecting game data (character animation data).

In STZ3, we modified the original VBA function that was used for forcing values of addresses (Cheat function) to be a tool for helping us to collect animation data. By writing a value from the start animation value to the end of animation value, we forced each character to do all of its actions. We captured the character's image of each action and saved it with its animation value as its file name. We then knew the animation value for each of a character's action. Human intervention was needed to identify the meaning of each action. An animation database was then produced.

4 EXAMPLE EXPERIMENT IN STZ3

This section will discuss the result of using AI-TEM with STZ3 ROM to create a simple, static script AI. Table 1 contains the addresses of STZ3 game state data that we know by the method discussed in section 2.0. Table 2 contains some character animations from a character named RYU, which we collected after normalization, by using stand animation as a basis. (Each animation is composed of many frames, so there are many values in each animation. Figure 4 shows some pictures of animations from table 2.

Table 1**:** Example address of STZ3 game state data.

Game State Data	Address	Data Size
character 1 position x axis	0x20007C2	16 bits
character 1 position y axis	0x20007C4	16 bits
character 2 position x axis	0x20043D2	16 bits
character 2 position y axis	0x20043D4	16 bits
character 1 Animation	0x20007D0	32 bits
character 2 Animation	0x20043E0	32 bits

Table 2**:** Example of character (RYU) animation.

Animation	Value
Stand	0, 12, 24, 36, 48, 60
Crouch	276, 288, 300, 312, 72, 228, 240, 252, 264
Jump	420, 432, 468, 480, 492, 504, 516, 528, 444, 456
Punch (Figure 4)	8484, 8496, 8508, 8520, 8532, 8544, 8556
Kick (Figure 4)	9096, 9108, 9120, 9132, 9144, 9156, 9168

4.1 AI Script Experiment

In order to test our AI module and python interface, we had implemented a static script to control a character in STZ3. Our test condition for our static script is the character RYU VS RYU in versus mode. This static script also allows us to test our AI in a controlled situation. We implemented a static script for character RYU. Our 1P's RYU can detect states of original game AI 2P's RYU.

Our first version of the script just randomly performs action. The result was not as bad as we originally believed. Even though it had no intelligence, it performed action continuously and was able to beat the original game AI at the easiest level. We improved our script in many aspects, using animation data that we collected. Our static AI can now sense distance between characters. We also script it not to use special moves often, since special moves leave characters defenseless. More combination attacks were also added. We obtain a better result, as expected. Our static AI can now beat the original game AI in middle level.

This experiment convinced us that AI can make use of the game state and animation of the opponent by using data in section 4. We also have a fully working python interface ready for creating future controlled situation.

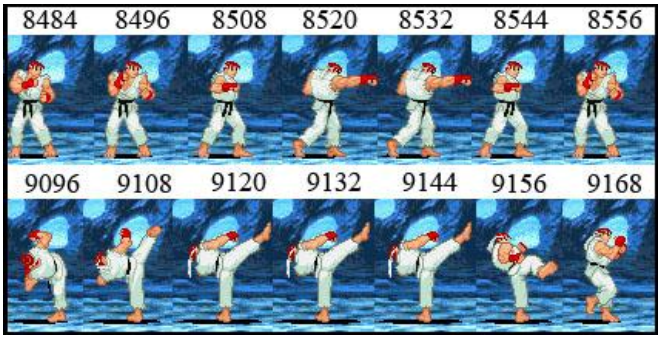

Figure 4: Example pictures of animation values.

5 DISCUSSION

5.1 Outline steps of Using AI-TEM

Researchers must first find a game that is suitable for testing their AI method or matches their experimental plan.

Researchers then identify the game states data that their AI module needs to know. In normal AI method, such as scripting, an AI module needs to know only current situations of the game. But in some AI method such as some type of Reinforcement Learning, it needs to know a complete set of actions that the agent can perform in every situation.

After that, they must find the address of game state data that their AI needs to know.

After the addresses are found, data must be collected from those addresses and translated into a form that the AI can understand.

Finally, AI can be implemented. This topic will be discussed in more detail in section 5.2.

5.2 Which AI method can AI-TEM be used with?

AI-TEM does not limit AI methods that it can be used with, because its concept is only using an emulator as game engine. Some AI methodology, however, requires extra functions. For example, using Genetic Algorithm requires running tests large amount of times, may be hundreds or thousands generation. Therefore, automatic result recorder is needed. High speed running mode will also be an additional welcome, for it can save time to train AI.

High speed mode is already available in VBA and many other emulators. Not all games may allow us to provide automatic running mode. This is because, if we cannot find memory data address that tells us about the beginning and

the end of the game, we cannot force the situation. But in general, automatic result collection can be done. Therefore, various AI techniques can be used.

5.3 What kind of Game/AI-Subject should use AI-TEM?

If researchers are interested in first person shooter, real-time strategies or D&D-style RPG game, there are games that come with tools. Good testing environment for such games can be built with such tools. Also, there are very few of these games on consoles. AI-TEM may not be the first choice for testing such games. If researchers are interested in simple platform action game, writing a game from scratch or finding some open source clone game is not a bad choice because all environments of the game can be fully controlled. However, developing games, from tools provided by a game, or from an open source clone cannot easily get us commercial-quality game. This is where AI-TEM can come in. AI-TEM can be used to test an AI developed on a simple, but fully controllable environment, against real commercial game. In the case of racing game, it is difficult to know game state data such as opponent car position and the track situation. As a result, AI-TEM will not be appropriate. For fighting game, we think that using AI-TEM is suitable, because this type of game is rather difficult to make and even more difficult to make it as good as commercial game. Therefore, we think the tradeoff in the case of fighting game is worthwhile. For sport game, we think that it is still suitable to use AI-TEM, because of the same reason as fighting game. Even though there may be many game states that an AI module needs to know, finding them may be easier than creating a high quality sport game from scratch. Some AI researchers use Robocup simulation league to be a testbed for their football AI research (Sean Luke 1997). However, Robocup simulation league rules are still not the same as real football rules.

For other types of games/AI-subjects, researchers have to consider the same factors as in this section.

5.4 The Limitation of using VBA in AI-TEM

To play a multiplayer mode in VBA (Link), two or more instances of emulators have to be used. Controlling many emulators at the same time while testing is not very convenient. It will be better if the second instance can run in the screen-off mode, in order to save CPU power.

Sometimes two connected emulators do not synchronize. This may damage the automatic module in long run. Detecting game state of both VBAs becomes necessary. We can then reload the game again if they do not synchronize.

Although there are some inconveniences, AI-TEM generally works well in our experiment. The emulator can be fixed to tackle the problem.

6 CONCLUSION AND FUTURE WORK

Our work provides an environment for testing AI on a wider range of commercial-quality games. Our experiment shows that, with appropriate game ROM, AI-TEM meets the three requirements in section 1 (test with a commercial-quality game, the game should not be too simple and there are many ways to play the game). Researchers can use AI-TEM to test their AI against the game's original AI or against a human opponent. Emulator players form huge communities, therefore many players can help with AI testing. AI developed by researchers can also be tested against AI running on script. A well designed script can help an evolutionary or learning AI improve in an appropriate direction. Although the GBA is not as powerful as next generation hardware, many games on GBA are regarded as classics and have been re-released on several new platforms. Therefore, AI-TEM is very much viable as testing environment for commercial-quality games. And the framework of AI-TEM should be adapted to more emulators in the future.

We plan to improve the implementation of the system and its associated tools. To provide a package for AI research in the future, we plan to collect the animation data of all characters in STZ3. We also want to build a cooperative AI for sport games using our testbed. The game WORLD SOCCER Winning Eleven is a perfect candidate ROM for the task. We also have plans to use another emulator with AI-TEM, such as MAME or ePSXe, in order to access more types of games. Multiplayer games can be run on MAME and ePSXe without synchronization or performance problems because only a single instance of emulator is needed.

REFERENCES

Aha, D.W., & Molineaux, M. 2004. Integrating learning in interactive gaming simulators. Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04). San Jose, CA: AAAI Press

Bailey, C. and M. J. Katchabaw. 2005. An Experimental Testbed to Enable Auto-Dynamic Difficulty in Modern Video Games. Proceedings of the 2005 GameOn North America Conference. Montreal, Canada.

Graepel Thore, Ralf Herbrich, Julian Gold. 2004. Learning to fight. International Conference on Computer Games: Artificial Intelligence, Design and Education.

Kendall Graham, Kristian Spoerer. 2004. Scripting the Game of Lemmings with a Genetic Algorithm. Proceedings of the 2004 Congress on Evolutionary Computation, IEEE Press, Piscataway, NJ, pp.117-124

Ponsen Marc J.V., Hector Munoz-Avila, Pieter Spronck, and David W. Aha. 2005. Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning. Proceedings The Twentieth National Conference on Artificial Intelligence.

Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, James Hendler. 1997. Co-Evolving Soccer Softbot Team Coordination with Genetic Programming. First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence.

Spronck Pieter, Ida Sprinkhuizen-Juyper, Eric Postma. 2004. Online Adaptation Of Game Opponent AI With Dynamic Scripting. International Journal of Intelligent Games and Simulation, Vol. 3, No. 1, University of Wolverhampton and EUROSIS, pp.45-53.

Python (2006). Python Language
http://www.python.org

VisualboyAdvance. (2005). GBA Emulator
http://vba.ngemu.com

VisualboyAdvance Link. (2005). GBA Emulator
http://vbalink.wz.cz/index.htm