|  |  |
| --- | --- |
| S | S17 |
| N |  |

****

**De La Salle University • College of Computer Studies**

**INTESYS / Introduction to Artificial Intelligence**

**AY 2014-15 Term 1**

**Major Course Output 1**

**A Generic Stealth Game Bot**

**(Design and Evaluation of its Performance)**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For students of Dr. Raymund Sison

Name (last name, first) : Aquino, Kurt Neil

Choy, Matthew Seaver

Hade, Alden Luc

Section : S17

E-mail address : kurt\_aquino@dlsu.edu.ph

Date of Submission : June 30, 2014

Time of Submission :

We hereby certify that we wrote our entire bot and report without help from anyone outside of our group.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Aquino, Kurt Choy, Seaver Hade, Alden

**Table of Contents:**

1. Introduction page 1
2. Artificial Intelligence Features page 1
3. Artificial Intelligence Improvement Recommendations page 2
4. Summary and Lessons Learned page 2
5. Appendix A. Contribution of Members page 3
6. References page 4

**Introduction**

\_Most video games which involve bot interaction would not be very entertaining nowadays without the application of artificial intelligence. Strategy games, like those that involve stealth, is a type of video game which relies on tactics, strategy, and quick-wit to overcome, or overpass, enemies and obstacles and is an example of a game in which many, if not all, find difficulty in completing.

\_But what if we wanted to create an AI, a bot, that could complete the game for us? That question led to the development of the GENERIC STEALTH GAME. And despite having a boring name, the term “generic” means more than just what it leads to. *Generic programming* is a computer programming model based on functions or classes defined regardless of the data types they initially have. An example of generic programming is using JavaScipt, where data types are not stated.

\_The Generic Stealth Game, or GSG, uses generic programming in order to complete a maze-style stealth game; and in order to do that, we created a bot that could find its way to the exit in the shortest time possible while also avoiding all enemies along the way.

\_The bot, nicknamed KemBot, because of the way it weaves back and forth through tight spots, uses the A\* search algorithm for its pathfinding and graph traversal, which is the process of generating an efficient path between nodes. It uses heuristics to achieve better performance. Some efficiency, in terms of speed, however, had to be sacrificed in the game since the bot does not only need to find the best path towards the nearest possible exit, but it also has to avoid being detected by enemies along the way.

**AI Features of KemBot**

\_The bot uses A\* search algorithm in order to find the most efficient path. This has been proven time and time again to be the most efficient and has widespread use.

\_We augmented the search algorithm to block tiles that have enemies on them, or will have enemies on them (enemy paths), this way we may not trip any alarms on the way to the exit.

\_The main object, TestBot(), consists of four functions: initAI(), thinkAI(), spottedAI(), and endAI().

* InitAI() is called at the start of every map. For here we implemented a short delay so that the enemy bots get to move first. This is so that the bot doesn’t crash into an enemy that hasn’t moved yet.
* ThinkAI() is called every .33 seconds. Compared to enemy bots, our bot moves five times faster. ThinkAI() is called to allow the bot at the current position to decide where to go next. It reviews enemy locations and their sight lines, and will actively avoid all of them and where they’re going on the way to the exit.
  + In this function, another big function, thinking(), is called. This is where the actual thinking and moving is done. Basically it does four things:
    1. If the end goal exists, get the heuristics (computeH(), computeG()) of the tiles from the current position to the goal;
    2. Based on these heuristics, get the best path (pathFind());
    3. If the end goal does not exist, look for another end goal and do 1 and 2 again;
    4. Act upon those decisions accordingly by doing player.moveUp(), moveDown(), etc.
  + With this function, the bot moves by A\* search but also avoids all hazards (isHazardous(), genHazards(), getEnemyArea(), inArea(), etc.)
* SpottedAI() is called *if* Kembot gets spotted despite our awesome algorithm, right before the user restarts the map and basically gives Kembot another chance. This is used for the AI to adapt and learn, to avoid making the same mistake, on the next attempt. We used this to avoid going into the same end goal (given a lot of end goals) if the AI gets caught on the way there too many times.
* EndAI() is an optional function; it is called by the main program whenever the AI gets to an end goal without getting spotted, right before the next map loads. We did not add anything to this and left it blank, although we could clearly see a good implementation of this.

**Artificial Intelligence Improvement Recommendations**

* Theoretically, endAI() can be used and programmed to augment the artificial intelligence after each success. Just like spottedAI() augments the AI by letting it see other possibilities, endAI() could also help the bot learn.
* Like, “Hey, I finished this level in three tries. Where did I go wrong and how do I go about avoiding making the same mistakes on the next level, so that spottedAI() gets to be called less and less and less?”
* Tailing an enemy bot usually leads to our AI rapidly weaving back and forth behind it while still keeping up with it, because it’s trying to search for another path to the end even though there already isn’t one. If this odd behavior can be eliminated, the AI would look much more realistic, which is what all games are aiming for.
* Concerning on how our bot backtracks in order to avoid encountering enemies, it would be better if it just stopped, and waited until the enemy passes or a clear path has been made.
* Rather than considering a new path and forgetting the path the bot previously took where it was captured, it would be better if the bot still considered the same path and waited for the enemy to pass.
* Overall, our bot is very one-track minded as it charges to the goal as soon as it generates a planned path, disregarding possible encounters with the enemy. It would be better if our bot manages to inch every best possible path, avoiding enemies whilst taking the time it takes into consideration, on the way to its goal.

**Summary and Lessons Learned**

\_As artificial intelligence’s (in general) advancement progresses with each passing day, so does the number of its applications to our everyday lives. From robots to programs, from entertainment to research, artificial intelligence will from now on, as it was before, be a key element in the advancement of our technology for a better living.

\_In our case, applying artificial intelligence to simple map traversal has become one of the more difficult programming outputs we have had so far. Our understanding with artificial intelligence back then was just simly random values and multiple if-else statements, and obviously, artificial intelligence is nothing of the sort. Artificial intelligence, as it is included in its name, is supposed to be intelligent; it must be aware of its surroundings, interact with its environment properly, perform its programmed task not only effectively but efficiently as well.

\_Although our task was to create an intelligent bot, we can only modify one to the best of our abilities and current knowledge, though it might not be intelligent at certain times, it still does its job. Compared to other bots, our bot has considered intentionally confronting an enemy, close enough to follow it, but not nough to be get caught. We assume that this is unique as other bots avoid the paths where the enemy lies and find an alternate route. Our bot disregards the first condition, and follows the enemy until it is able to search for another route. The only thing wrong with this if the bot follows a certain path and gets caught at the end multiple times, it disregards the entire path itself and looks for another one. In a sense, this is not intelligent as itn does not first consider all possibilities before proceeding to an alternate path. This in turn will cost our bot traversal time, as well as the number of times it retries the stage.

\_An intelligent bot does not only consider all of the possible paths but the locations of the enmy bots as well during specific time lapses, or in this subjects terminology, an intelligent bot must consider all possible states a map can have and take into account the shortest possible route to the goal. An intellogent bot does not rush towards the goal once a path has been set, no matter how quick it is. Having a slow and steady bot, avoiding enemies, traversing though the shortest path, backtracking to previous paths when cornered, reaching the goal in the shortest time, is the ideal bot. Half of which is what our bot is only capable of doing. We believe that with further study and analyzation we may be able to program a bot close to the ideal one but never exactly, as more difficult obstacles can be placed in certain maps in which our bot may not be able to handle any further.

**Appendix A: Contribution of Members**

We hereby certify that the detailed contributions, contribution percentages, and time recording logs presented below are correct representations of the individual effort and contribution of each member.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Aquino, Kurt Choy, Seaver Hade, Alden

**Aquino, Kurt – 33.3%**

* Coded basic functions for traversing through the map
* Added functions for selecting multiple path
* genPath(closed, start, end)
* pathFind(start, end, maplayout, enemies)
* inClosed(point, closed)
* inOpen(point, open)
* Coded the heuristics algorithm (based on the referenced sites)
* computeF(g, h)
* computeG(parent, child, enemies)
* computeH(point, end)
* Code testing
* Debugging

**Choy, Seaver – 33.3%**

* Coded functions for detecting enemy tiles
* getEnemyArea(enemies, area)
* inArea(area, place)
* genHazards(enemies, enemyArea)
* isHazardous(player, enemies)
* Coded the actual thinking of the AI when spotted, selecting a path, getting the best path
* Thinking=function(player, enemies, maplayout, end)
* thinkAI= function(player, enemies, maplayout, end)
* spottedAI = function()
* Code testing
* Debugging

**Hade, Alden – 33.3%**

* General research for heuristics and A\* search
* Coded the intial programming of the bot
* initAI = function(player, enemies, maplayout, end)
* Coded the integration of the heuristics algorithm to the path generation functions
* tile(coord, parent, h, g, f)
* Coded the functions for queuing of the array of paths
* arrange(open)
* checkChange(parent, point, g, open)
* reverse(path)
* Code testing
* Debugging
* Commenting (in code)
* Code readability

**References**

Lester, P. (2005, July 18). A\* Pathfinding for Beginners. *A\* Pathfinding for Beginners*. Retrieved June 29, 2014, from <http://www.policyalmanac.org/games/aStarTutorial.htm>

Blue, A. (n.d.). A\* Pathfinding Demo. *A Star JavaScript Pathfinding Demo*. Retrieved June 29, 2014, from <http://ashblue.github.io/javascript-pathfinding/>

WenderLich, R. (2011, September 29). Introduction to A\* Pathfinding | Ray Wenderlich.*Ray Wenderlich*. Retrieved June 29, 2014, from <http://www.raywenderlich.com/4946/introduction-to-a-pathfinding>

Grinstead, B. (2009, June 3). A\* Search Algorithm in JavaScript. *Brian Grinstead RSS*. Retrieved June 29, 2014, from <http://www.briangrinstead.com/blog/astar-search-algorithm-in-javascript>

Grinstead, B. (2009, July 3). javascript-astar. *Demo*. Retrieved June 29, 2014, from http://bgrins.github.io/javascript-astar/demo/

Name (last name, first) : Aquino, Kurt Neil

Choy, Matthew Seaver

Hade, Alden Luc

Section : S17

E-mail address : kurt\_aquino@dlsu.edu.ph

Date of Submission : June 30, 2014

Time of Submission : 11:00 – 12:30 (class time)