NC STATE UNIVERSITY

Study of decision making and path planning algorithms with an AI game

Anshul Jethvani
Raj Kumar Shrivastava
Tanay Agrawal

Problem statement

Analysis and comparison of algorithms used for path finding and decision-making

Approach - General Idea

We have selected the following algorithms in our study:

- Path-finding
  - Recursive best first search algorithm
  - A-star

- Decision-making
  - Goal-oriented action planning with IDA*
  - Behavior trees

- Tools:
  - Processing

- Language:
  - Python

Approach - Game Elements

We have designed a game to experiment with the algorithms being used in our study.

- The game consists of following:
  - A human player
  - Two guards (AI bots)
  - A treasure
  - Obstacles
    - Deadly fire
    - Walls
    - Other objects
  - Power ups
    - SpeedUp: Increase player speed
    - SpeedDown: Decrease guard speed
    - Immunity: Ability to walk through the obstacles

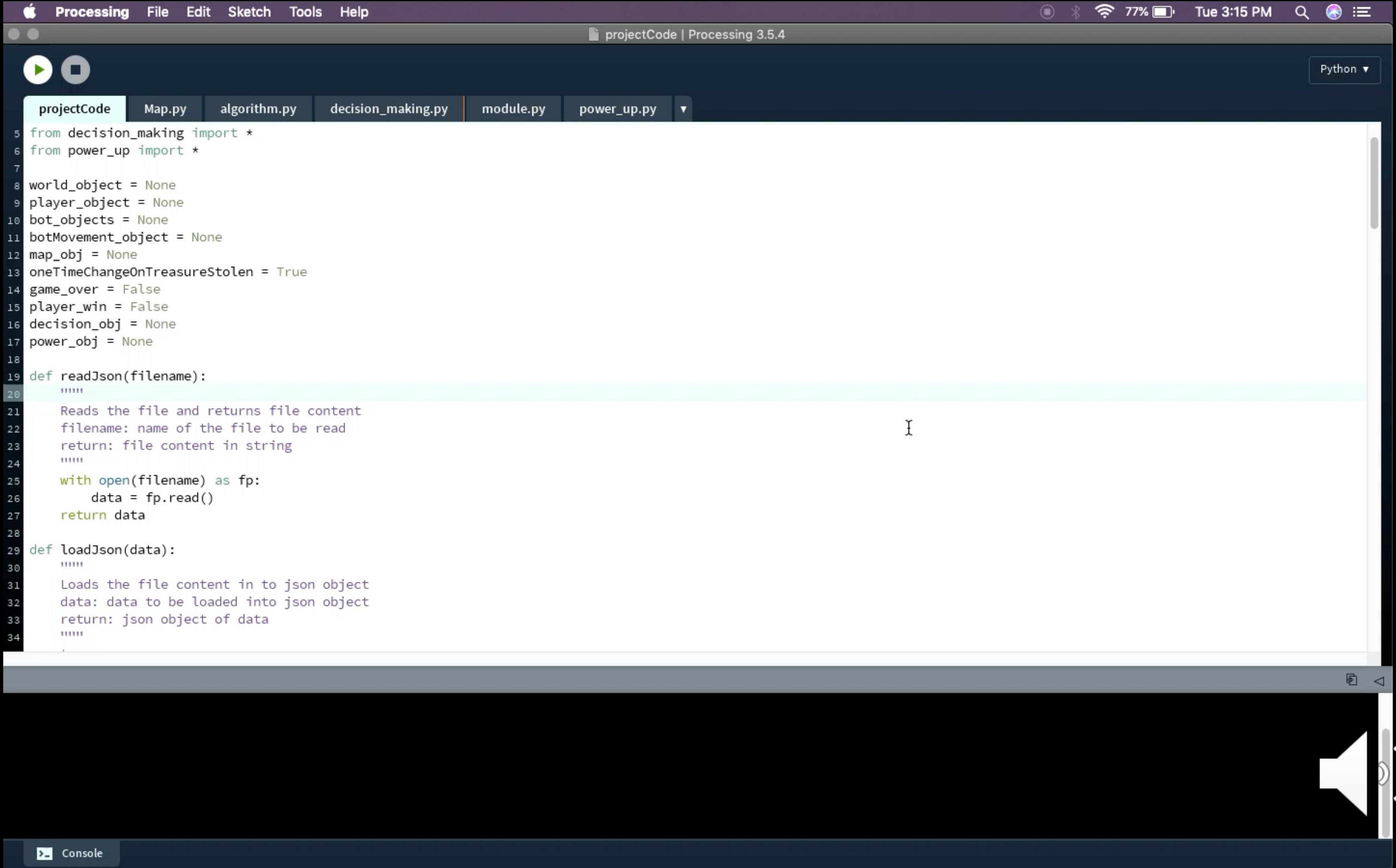
Approach - Game Design

- The game begins with the player starting at the safe house (top-left corner) and the treasure at the haunted house (bottom-right corner)
- Two guards are present whose action is decided by the decision-making algorithms
- The goal of the user is to collect the treasure from the haunted house and take it back to the safe house
- The challenge is to complete the task while avoiding collisions with the deadly obstacles and the guards.
- The player can also use several power ups that pop throughout the course of the game.

Approach - Bot Behavior

We implement our algorithms for the movements and decisions taken by the guards. The guards exhibit various behaviors:

- Wandering across the map
- Chasing the player
- Guarding the treasure (before treasure is stolen)
- Guarding the safe house (after the treasure is stolen)
- Guarding the power ups

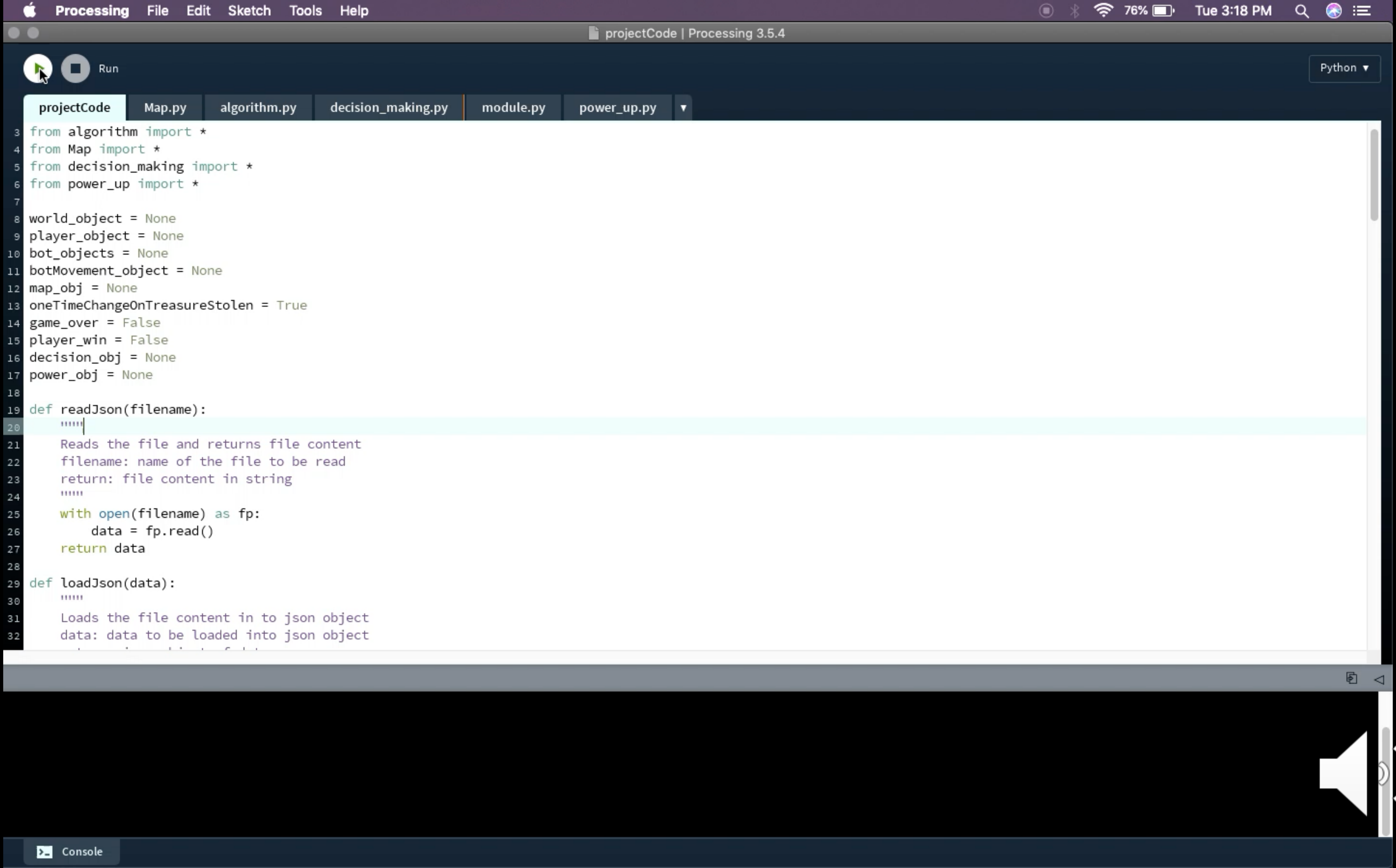


```python
from decision_making import *
from power_up import *

world_object = None
player_object = None
bot_objects = None
botMovement_object = None
map_obj = None
oneTimeChangeOnTreasureStolen = True
game_over = False
player_win = False
decision_obj = None
power_obj = None

def readJson(filename):
    """
    Reads the file and returns file content
    filename: name of the file to be read
    return: file content in string
    """
    with open(filename) as fp:
        data = fp.read()
    return data

def loadJson(data):
    """
    Loads the file content in to json object
    data: data to be loaded into json object
    return: json object of data
    """
```



```
from algorithm import *
from Map import *
from decision_making import *
from power_up import *

world_object = None
player_object = None
bot_objects = None
botMovement_object = None
map_obj = None
oneTimeChangeOnTreasureStolen = True
game_over = False
player_win = False
decision_obj = None
power_obj = None

def readJson(filename):
    """
    Reads the file and returns file content
    filename: name of the file to be read
    return: file content in string
    """
    with open(filename) as fp:
        data = fp.read()
    return data

def loadJson(data):
    """
    Loads the file content in to json object
    data: data to be loaded into json object
```

Project design and iteration

Key challenges, roadblocks and solutions

- Path finding for multiple agents simultaneously at every instant bogged down the processor. To address this problem, we improved the algorithm implementation and simplified the heuristics. We also the reduced the number of agents to 2.

- Implementing realistic collision detection with the key locations, guards and obstacles posed a problem due to the unaligned centre of objects.

- Defining the insistence values in case of GOAP and probability values in case of Behavior Trees, for multiple actions in various states and incorporating them in the decision-making process was a challenge.

- We also had to scale down the number of power ups and the UI of the game to reduce glitches and achieve a smoother game play.

Project design and iteration

We have created a different class for every entity in the game and the class encompasses corresponding attributes for that entity:

Player:

- Speed
- Current_location
- Immunity
- draw_player()

Bot:

- Speed
- Current_location
- Path_traversing
- Is_moving
- Destination
- Current_state
- draw_bot()

Bot_Movement:

- Bot_actions
- Bot_locations
- find_bot_path()
- update_bot_probilities()
- move_bot_decisions()

Map:

- drawMap()
- checkTreasureStolen()
- collision_detection()
- activate_powerUp()
- activate_powerDown()

Project design and iteration

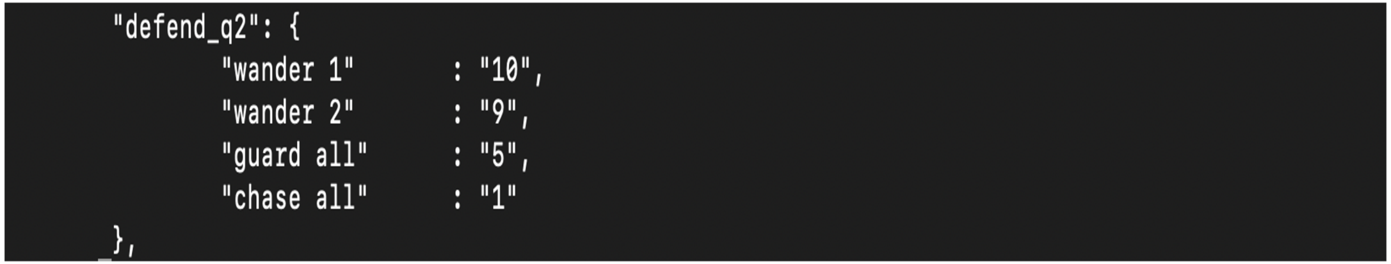
We have created a different class for every entity in the game and the class encompasses corresponding attributes for that entity:

PathFinder:

- pathFindAstar()
- rBFS_utility()
- pathFind_RecursiveBestFirstSearch()
- checkValid_location()
- heuristic()

Decisions():

- determine_state()
- heuristic_select_actions()
- get_goap_actions_list()
- get_behavior_tree_actions_list()
- game_control()



```
"defend_q2": {
        "wander 1"      : "10",
        "wander 2"      : "9",
        "guard all"     : "5",
        "chase all"     : "1"
},
```

Project design and iteration

Heuristic function for path finding and decision making algorithms

```
def heuristic(self, first_location, second_location):
    """
    Function that calculates the Eucledian distance
    between two points a and b on canvas
    return: the distance between the two points
    """
    return math.sqrt(pow(first_location[0] - second_location[0],2) +
                     pow(first_location[1] - second_location[1],2))
```

Filling the heap structure with actions using heuristics and insistence values

```
for temp_key in action_dict.keys():
    if self.current_game_state in self.defend_game_states:
        temp_f_value = self.heuristic(self.player_position, self.map_obj.data["key_locations"]["treasure"])
    else:
        temp_f_value = self.heuristic(self.player_position, self.map_obj.data["key_locations"]["safe_house"])
    heappush(actions_heap, (-1 * (int(action_dict[temp_key]) + temp_f_value), temp_key))
```

Project design and iteration

- We have standard sequencer and selector. But we have customized decorators.
- Decorator usually have only one child node.
- We have modified the Decorator to have multiple children but during execution only one of them is selected based on their respective probability and a random number generation.
- Thus we have a kind of Non-Deterministic Random Selector.

Experimental design

Experimental Hypothesis:

- Recursive best first search consumes less memory as compared to the A-star search algorithm which is an essential factor in game development in termps of path-finding
- Behavior Tree runs faster than GOAP IDA*.

Experimental design

Evaluation Parameters:

What did we measure?

- Theoretical Time and Space Complexity of the algorithms:

  b is the branching factor and m is max depth
  - **A***: O(b^m) for time and O(b^m) for space
  - **RBFS**: O(bm) for time and O(bm) for space (might differ in different case)

    n = max number of actions in every state,
  - **Behavior Tree**: O(n) in time and O(n) in space complexity
  - **GOAP IDA***: O(nlogn) in time and O(n) in space complexity

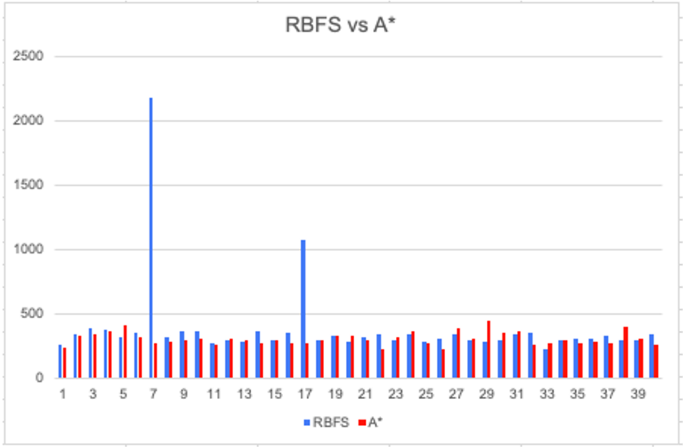
How did we measure?

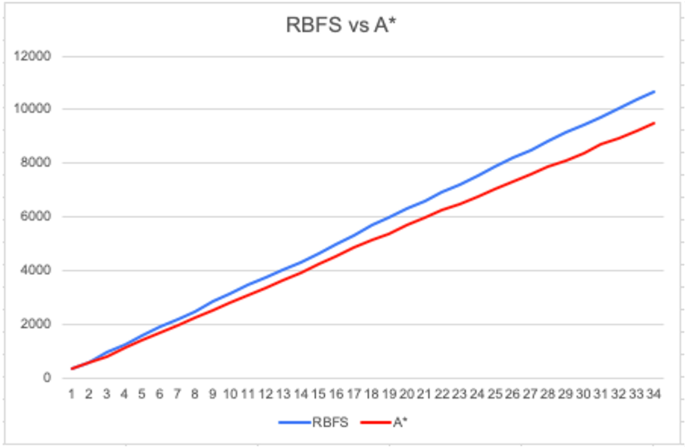
- The running time of the function to give the respective output using time library in python.
- The memory usage of the function by finding the maximum queue size while running the algorithm.

Results

Experimental results on Time complexity of the path finding algorithms implemented in the game:

- On a cumulative basis we can see that the time taken by Recursive best first search is more than A* search algorithm

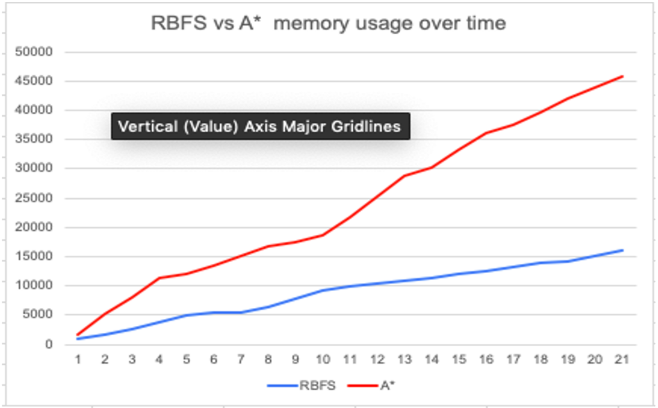


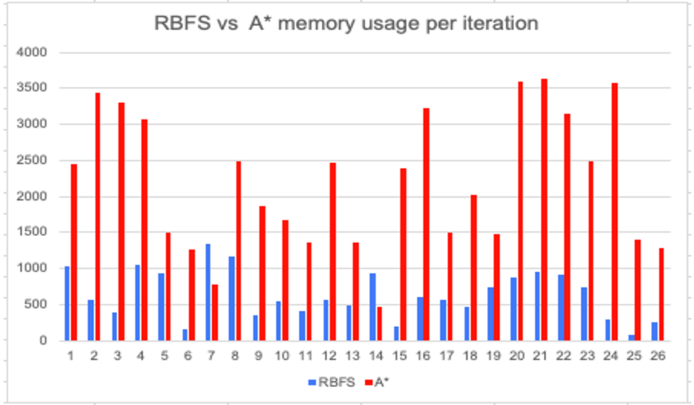


Results

Experimental results on Space complexity of the path finding algorithms implemented in the game:

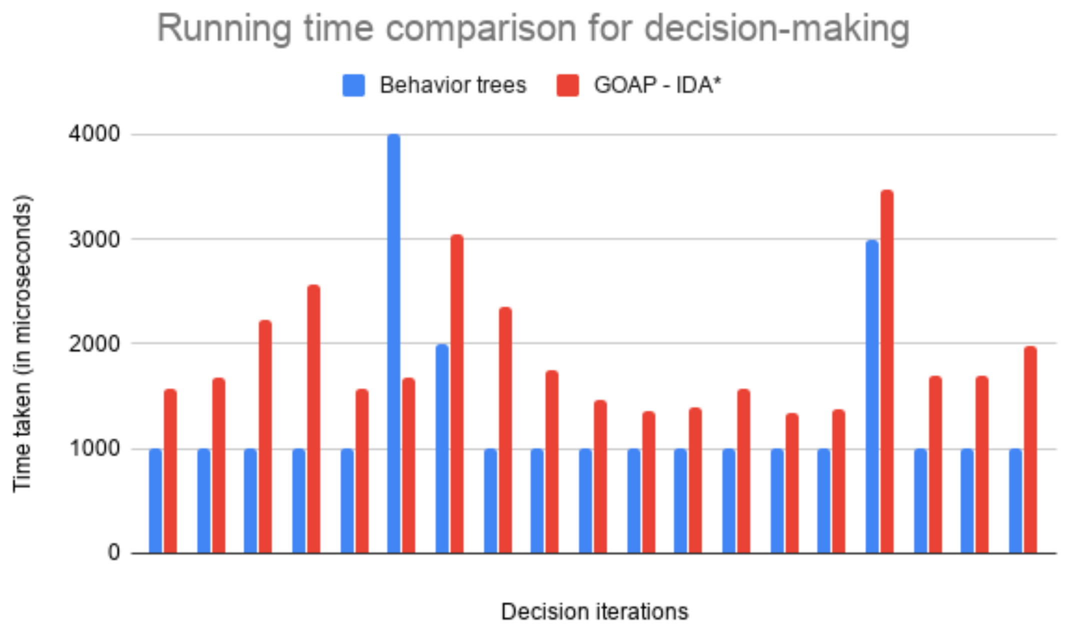
- The memory consumption of Recursive breadth first search (RBFS) algorithm is linear and less when compared to that of A* search algorithm.
- Cumulative memory growth of RBFS is less that of A* search algorithm.





Results

Experimental results on Time complexity of the Decision Making algorithms implemented in the game:



Conclusions

- In this work, we studied and analysed the behaviors of Recursive best first search algorithm and A-star algorithm to evaluate the performance in the game scenario.

- For decision-making, we focused on Goal-oriented action planning with IDA* and behavior trees for bot's decision-making to choose from various possible actions.

- The AI bots play against the human player in this zero-sum game. The goal of the human player would be achieved to steal the treasure back to the safe and decide whether to pick powers or not. And bots need to prevent this.

- This provided us a platform to evaluate the behavior of the bots in multiple scenarios to best fathom the pros and cons of the algorithms.

Conclusions

- The project helped us study and analyze the best use cases of different algorithms suited for different tasks like pathfinding and decision making.
- By using algorithms with different efficiencies, we were able to compare their theoretical and experimental run time and memory usage.
- With this project, we found the trade offs between user experience and the computational costs involved in execution of these algorithms.
- Innovating ideas with relatively less complex implementations can provide a great gaming experience if programmed the right way.

NC STATE UNIVERSITY

Any questions or further discussions are welcome.

Please direct your thoughts to any one of us and we would be happy to discuss.

- Anshul Jethvani (ajethva@ncsu.edu)
- Raj Kumar Shrivastava (rkshriv@ncsu.edu)
- Tanay Agrawal (tagrawa@ncsu.edu)

We welcome you to review our demo video, project code and report and would be glad to hear your feedback: https://github.com/amj23897/CSC584-Project

Thank you