INTRODUCTION:

Berzerk is a multi-directional shooter arcade game, released in 1980 by Stern Electronics of Chicago. Berzerk places the player in series of top-down, maze-like rooms containing armed robots. Berzerk is a one player game. The main challenge In Berzerk game is to score as many points as possible without being destroyed yourself. Each maze ends when the agent perishes or escapes. Initially, robots move and shoots slower than the agent. But with each consecutive maze, the robot's movements and firing speeds gradually increases. At Maze 16, the robots reset to move slowly but the firing speed remains the same as our agent. You will lose a life if you:

1. You are hit by a robot's laser.
2. Run into the electrified wall of maze.

In addition to three lives, extra lives can be won after level 6. In some levels, the robots aren't armed. You can stand in front of them, but they can't shoot you unless you touch them.

Your score is determined by the number of robots that are destroyed. Whether you shoot a robot, or it is eliminated by other means, you receive points for every pulverized robot. You will earn bonus points when all the robots in a single maze are annihilated. When all the robots in a maze are destroyed, your score disappears your bonus points flash on your screen. The computer automatically combines all points and your total score is displayed in the next maze.

METHODOLOGY:

We have designed 3 Algorithms quite distinct in nature but based on the same underlying agenda to maximize the score thus achieved by the provided base random agent.
The idea behind the Algorithms was based on the evolution principle of the base case agent.
The algorithms we have implemented have evolved themselves from the base case random agent itself.

Let's understand how we evolved our agents by introducing each one of them independently in this section.

Agent#1: Semi Intelligent Randomized Agent

This Agent is a buffed version to the base provided random agent in class.
The idea behind this agent is to smartly perform random actions, the actions are categorized into different sets each containing a unique probability to itself. Eg: The set for diagonal shooting can contribute to 60% of the in-game action where the robot will mostly shoot diagonally and walk limited.
This allows the agent more in-game time than the random agent provided which perishes early by running into a wall most frequently.

The Agent#1 is also empowered with some relay event of the game via Color codes of the enemies present in the line of action of the agent on which if found the agent terminates the random sequence and wishes to eliminate the target perceived in front of him first and then continue randomizing its action sequences.

Pseudo Algorithm:

- If the enemy is in sight of the agent (Up, Down, Left, Right)

  - If there is any wall in between -> **Don’t Shoot But don’t move either**
  - If there is no wall in between -> **Shoot based on the position of the enemy**
- Else
  - Randomize the action based on the following principle
  - Random.choice – choses an element from the array based on the probabilities provided to the action sequence in the array
  - Here each action set in the array possess the same probability for the action set to be performed. Eg: **If shooting is 40% then (10,11,12,13) each possess .4** as their respective probability in the array and any of them can be pulled at random when the choice is returned as .4
- Perform the above steps till all the enemies are not terminated via shooting or have themselves killed due to walls Or the agent kills itself
- If the enemies are killed and the agent is alive take the exit

Agent#2 Simple Reflex Agent:

This Agent is a buffed version to the Semi Intelligent Random Agent mentioned above.

The idea behind this agent came into existence only because of the shortcomings of the Semi Intelligent Random Agent which is that it still had a marginally high probability to get itself killed by moving randomly via touching walls or by getting killed via some enemy bullets in the later stages of the game.

The idea behind this is to smartly eliminate the enemies which are only in its line of action (Up, Down, Left, Right) with considering if the walls are in between the Agent and the enemy.
This allows the agent maximum game time.
How you wonder?
Answer: The game is strategically designed with the idea that the agent is the most powerful of them all, this being said the agent is the only element of the game which could shoot in all directions possible – Up, down, left, right and the respective diagonals. However the enemies are not that smart coded the enemies however smart enough to gauge the distance between them and the agent and to see that if they fire and the agent doesn’t terminate they walk away – far to allow the relay of their firing sequence to kill the robot in the delay of shooting.

This successively allows them to get themselves killed by touching the walls as they are blind coded with the information of the walls. The agent smartly uses this advantage to its favor by waiting at the position it was invoked and shoot anyone possible in its line of action to kill it.
Once everyone is dead and the agent survives it take the exit.

Pseudo Algorithm:

- If the enemies are in sight (Up, Down, Left, Right)
  - If the walls are not in between -> **Shoot**
- Else, **Do nothing**

- Perform the above steps till all the enemies are not terminated via shooting or have themselves killed due to walls Or the agent kills itself
- If the enemies are killed and the agent is alive take the exit

Agent#3: Real World Agent (Looks like a human child is playing)

This is the last and final version update of the agent we have developed over the iterations to produce something when played looks like to any third party that some human is playing, preferably a child because this unlike the other two still acts dumb at some instances of the game which can refer enough to a small child with half knowledge of the game.

The idea behind this agent came into existence only because of the high order of running time for the Simple Reflex agent at various intermediate levels where the agent seemingly just waits for the enemy to kill itself with a wall from which it is initially quite far, the enemy fires nearly 250-300 bullets before it starts walking and kills itself.

To eradicate this feature from the game we decided to make something which perceiving the environment tries to follow the enemy and kill it, unlike waiting at one position and make the game seemingly boring, however there are always a tradeoff between feel and results the later proved to be a bit costly as the score lowered considerably while behaving quite humanly.

The approach for this agent is pretty much how a human perceives a game, find the nearest enemy get it's location and then dispatch itself on a mission to kill the designated enemy location just like how a human would like a game to be, anyone seeing this game from one side of the mirror could maybe at times not guarantee that its not a human playing this version.

Let's get to the algorithm for this one,

Pseudo Algorithm:

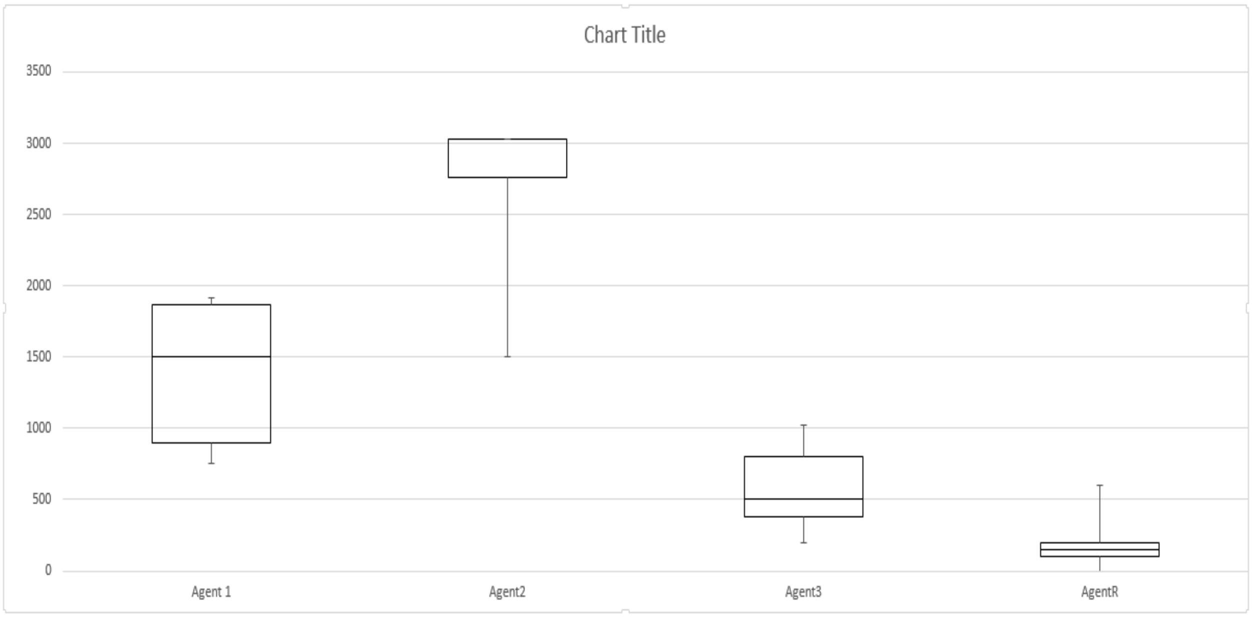
- Scan the environment completely
- Store each and every location of the enemy at each frame instance
- Run the FindNearestEnemy Function to find out the nearest enemy
- Check what is the best route to reach the enemy in-order to kill it
- Check the follRow and follColl approach
- Once in sight – Kill the enemy, the above approach guarantee's no presence of wall at this place
- Once eliminated, follow steps 3 – 6 again and again till all enemies are not dead
- Perform the above steps till all the enemies are not terminated via shooting or have themselves killed due to walls Or the agent kills itself
- If the enemies are killed and the agent is alive take the exit

The only drawback of this algorithm is the seemingly same color of the enemy and its bullet which leads the agent to believe the bullet too being one of the enemy and tries to eliminate it in one way trying to save itself and the other way gets itself killed in trying to adjust itself in line with the bullet to try eliminate it, also when multiple enemies shoot bullets, it becomes quite complex for it figure which bullet (Enemy) is near enough to be eliminated and doing this it could get killed by the other bullet, However this approach reduces the optimized score but this still provides a more acting humanly feeling which is something we are trying to achieve even if the score has a dip between approaches still proving to be quite more efficient than the very base random agent provided.

RESULTS:

The aim for this project is reject the null hypothesis that the base case random agent is able to outperform any other algorithm developed by us.
In our claim to reject this hypothesis, we have collaborated evidences on various fronts, primary being the box and whiskers plot to help show the max-min and median performance of each of the three agents with the distribution generated from the random agent thus received on a trial of over 80 runs.

The box and whiskers plot:



From the chart above we can draw some conclusion like:

Agent1 – Semi Intelligent Random Agent:

- Mean = 583
- Max = 1020
- Min = 200

Agent2 – Simple Reflex Agent

- Mean = 2834
- Max = 3030
- Min = 1500

Agent3 – Real world Agent

- Mean = 1407
- Max = 1870
- Min = 750

AgentR – Random Agent provided

- Mean = 158
- Max = 600
- Min = 0

Clearly all the above executed algorithms perform better than the base case random agent, however let us perform the T-Test on the data.

The T-Test hypothesis claim:
Agent1 compared with AgentR:
- t=14.6
  sdev=212.
  degrees of freedom =200
- The probability of this result, assuming the null hypothesis, is less than .0001

Agent2 compared with AgentR:
- t=77.5
  sdev=245.
  degrees of freedom =200
- The probability of this result, assuming the null hypothesis, is less than .0001

Agent3 compared with AgentR:
- t=27.0
  sdev=329.
  degrees of freedom =200
- The probability of this result, assuming the null hypothesis, is less than .0001

Conclusions are made later in the document.

DISCUSSION:

Parameters	Agent#1	Agent#2	Agent#3	AgentR
Bullets fired	56	62	50	20
Total Bonus	50	420	220	0
Enemies Killed	4	25	26	2
Life Increased	0	2	1	0
Time's defended itself from the bullet	0	11	20	0
Enemies killed on own	6	22	5	0
Percentage map covered	10%	15%	30%	5%
Levels cleared and exited	1	12	3	0
Agent suicide	3	0	1	3
Total Time	2min 30 Sec	40min	25min 30 Sec	20 Sec
Envionrment seen	4	14	7	3
Time's enemy try to kill agent	6	15	25	2
Agent killed by enemy bullet	4	2	1	0
Enemies killed with Agent suicide	0	0	2	0
Score	550	3030*	1870	100

Quality Inference on Bullets fired to enemies killed:

Efficiency% On shots fired	7.142857143	40.32258065	52	10

The above data compilation is produced by running each algorithm once, and have tabulated some nuance information which could further help understand the better working of each algorithm, this analysis helps in understanding one the need of utilization on a particular algorithm based on what kind of result one wishes to achieve.
For Eg: If I would like to participate in any competition where the time taken to complete the game requires the second highest weightage and score being the primary factor I can decide which of the above 4 algorithms proves to be more efficient in my approach. Being the 1ST one. Such novel results can be mined further from the above running inferences from the above results.

CONCLUSION:

We conclude from various supports made towards our claim that we can reject the null hypothesis and claim that the random agent provided can nowhere near defeat any of our prepared algorithms.

FUTURE SCOPE:

The real world agent can be further upped by adding the constraints of defending itself from the bullets better, this would allow better running performances.