****

**Faculty of Informatics and Computer Science**

**Logic and Artificial Intelligence**

12) AI AGENT FOR BATTLE CITY

A project proposal for the Logic and Artificial Intelligence

***Presented by***

*Ahmed Ramie*

*Mariam Salah*

*Mohamed Hany*

**BUE, El-Sherouk City, Cairo, Egypt**

03/2023

# Project Members:

|  |  |  |
| --- | --- | --- |
| Student Name | ID | Each student’s responsibility in the project |
| Mariam Salah | 196292 | Collection of Data and Pre-Processing Data. |
| Ahmed Ramie | 217380 | Building up the AI model, Adding Algorithms to the desired application. |
| Mohamed Hany | 206457 | Testing , validating and visualize the model behavior. |

# Problem Description and Background:

**Problem Description:**

Battle City is a classic video game in which players control a tank and attempt to destroy enemy tanks while defending their own base. The game has been popular since its release in 1985 and is still played by gamers today. While the game is simple in concept, it presents several challenges that make it an interesting problem for AI research. Specifically, developing an AI agent that can effectively navigate the game's terrain, avoid obstacles, and make strategic decisions in real time is a difficult task.

**Background:**

In recent years, there has been a significant increase in the development of AI agents that can play video games. These agents are typically built using machine learning techniques, such as reinforcement learning, and can learn to play games at a superhuman level with enough training data. However, developing an AI agent for Battle City presents a unique challenge due to the game's complex and dynamic environment.

One of the main challenges in developing an AI agent for Battle City is the need to navigate the game's terrain in real time. The game features a wide variety of obstacles, including walls, trees, and water, which the agent must be able to navigate around. Additionally, the agent must be able to make strategic decisions on the fly, such as whether to attack an enemy tank or defend its own base.

Another challenge in developing an AI agent for Battle City is the need to balance exploration and exploitation. The agent must be able to explore the game's environment and learn from its experiences, while also exploiting what it has learned to make strategic decisions in the game.

Overall, developing an AI agent for Battle City presents a unique and interesting challenge for AI researchers. By successfully developing an AI agent for this game, we can gain insights into how to build agents that can effectively navigate complex and dynamic environments, make strategic decisions in real time, and balance exploration and exploitation.

# Software and Tools used to accomplish this project:

1. **Python:** Python is a popular programming language used in machine learning and AI development. It has a variety of libraries and frameworks, such as TensorFlow, PyTorch, and Keras, which can be used to build and train machine learning models.
2. **OpenAI** Gym: OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It includes a variety of environments, including classic Atari games like Battle City, that can be used to test and train AI agents.
3. **Jupyter Notebook**: Jupyter Notebook is an interactive web-based tool for writing and running code in Python. It can be used to experiment with different machine learning models, visualize data, and document the research process.
4. **TensorFlow:** TensorFlow is an open-source machine learning library developed by Google. It can be used to build and train deep learning models, including neural networks, for various applications, including game AI.
5. **PyTorch:** PyTorch is another open-source machine learning library that can be used to build and train deep learning models. It offers dynamic computation graphs and is popular among researchers and developers alike.

# Resources found (books, tutorials, etc.):

1. **Reinforcement Learning:** An Introduction (book) - This book by Richard Sutton and Andrew Barto provides a comprehensive introduction to reinforcement learning, which is the machine learning technique often used for developing game AI.
2. **Deep Reinforcement Learning (tutorial)** - This tutorial by Sergey Levine provides an overview of deep reinforcement learning and how it can be used for game AI development.
3. **PyTorch Tutorials (tutorials)** - The PyTorch website provides a variety of tutorials and examples that can help developers learn how to use the PyTorch library for deep learning and reinforcement learning.
4. **TensorFlow Tutorials (tutorials)** - The TensorFlow website provides a variety of tutorials and examples that can help developers learn how to use the TensorFlow library for deep learning and reinforcement learning.
5. **OpenAI Gym Documentation** (documentation) - The OpenAI Gym website provides comprehensive documentation and examples for using the OpenAI Gym toolkit for developing and testing reinforcement learning algorithms.
6. **Atari Learning Environment (ALE) (toolkit)** - The ALE is a toolkit for developing and testing reinforcement learning algorithms on classic Atari games, including Battle City. It provides a standardized interface for interacting with the game environment and collecting data for training AI agents.
7. **GitHub** is a code hosting platform that provides version control, collaboration tools, and various resources for developers to share and contribute to open-source projects.
   1. **Code sharing:** You can use GitHub to share code with other team members, collaborate on the code, and track changes.
   2. **Open-source libraries and projects**: GitHub hosts a wide range of open-source libraries and projects related to AI, machine learning, and game development that can be used as references or even integrated into your project.

PEAS

* **Performance Measure**

1. **Number of enemy tanks destroyed:** This measure evaluates the agent's ability to eliminate enemy tanks and reduce their overall number on the game board.
2. **Time taken to complete a level:** This measure evaluates the agent's efficiency in completing a level.
3. **Health points remaining**: This measure evaluates the agent's ability to preserve its own health and avoid getting destroyed.
4. **Score**: This measure evaluates the agent's ability to earn points by destroying enemy tanks, power-ups, and completing levels.
5. **Number of power-ups collected**: This measure evaluates the agent's ability to collect and utilize power-ups, which can enhance its abilities and increase its chances of survival.

* **Environment**

1. **Map layout**: The layout of the map where the game takes place, including walls, obstacles, and power-ups.
2. **Enemies**: The number and type of enemies that the agent must face, including their behavior and attack patterns.
3. **Terrain**: The different types of terrain that the agent must navigate through, such as water, ice, or sand.
4. **Power-ups:** The different power-ups that can spawn on the map, including speed boost, invincibility, and increased firepower.
5. **Time**: The time limit for each level and the remaining time the agent has to complete its objectives.

* **Actuators**

1. **Movement controls**: The AI agent can use the actuators to control the movement of the tank in different directions such as up, down, left, and right.
2. **Shooting mechanism**: The AI agent can use the actuators to shoot at the enemy tanks, destroying them to progress through the game.
3. **Defense mechanism**: The AI agent can use the actuators to deploy defensive measures such as shields or walls to protect itself from enemy fire.
4. **Power-ups collection**: The AI agent can use the actuators to collect power-ups such as extra lives, increased firepower, or increased speed to improve its chances of success.
5. **Communication with team members**: If the game allows team play, the AI agent can use the actuators to communicate with its team members to coordinate attacks and defensive strategies.

* **Sensors**

1. **Camera sensor:** to perceive the surroundings of the tank and detect enemy tanks and obstacles.
2. **Radar sensor**: to detect and track incoming projectiles from enemy tanks.
3. **GPS sensor**: to locate the position of the tank on the map and navigate to specific destinations.
4. **Health sensor**: to monitor the health of the tank and detect any damage or need for repair.
5. **Communication sensor**: to receive messages from other tanks or the central command and update the agent's decision-making process.

PAGE

* **Preceptors**

1. **Enemy detection:** This preceptor would allow the AI agent to detect and track enemy tanks on the map. It could be achieved using image recognition techniques to identify enemy tanks based on their appearance.
2. **Obstacle avoidance**: This preceptor would enable the AI agent to avoid obstacles on the map, such as walls and trees. It could use techniques such as pathfinding and collision detection to determine the safest route through the map.
3. **Power-up collection:** This preceptor would allow the AI agent to collect power-ups scattered throughout the map, such as extra lives and improved weapons. It could use reinforcement learning techniques to determine the optimal strategy for collecting power-ups.
4. **Base defense:** This preceptor would enable the AI agent to defend its base from enemy attacks. It could use techniques such as rule-based systems to determine when to engage enemy tanks and when to retreat to a defensive position.
5. **Strategic decision-making:** This preceptor would allow the AI agent to make strategic decisions about how to best achieve its objectives in the game, such as whether to focus on attacking enemy tanks or defending its base. It could use machine learning techniques such as decision trees or neural networks to learn from past gameplay data and make better decisions over time.

**Actions:**

1. **Move:** The agent can move in one of four directions - up, down, left, or right.
2. **Shoot**: The agent can shoot a bullet in the direction it is facing to destroy enemy tanks or obstacles.
3. **Collect power-ups:** The agent can move to and collect power-ups such as shields, extra lives, and faster bullets.
4. **Avoid enemy fire**: The agent can move to avoid enemy bullets and prevent itself from being destroyed.
5. **Target enemy tanks**: The agent can move and shoot at enemy tanks to destroy them and prevent them from reaching the agent's base.

* **Goals**

1. **Survive**: The primary goal of the AI agent would be to survive for as long as possible. This would involve avoiding enemy fire, navigating around obstacles, and finding power-ups to boost the agent's health and abilities.
2. **Destroy enemy tanks**: Another key goal would be to destroy as many enemy tanks as possible. This would involve identifying and prioritizing targets, positioning the agent to attack effectively, and using different types of ammunition to maximize damage.
3. **Protect base:** The agent could also be programmed to protect its base from enemy tanks. This would involve positioning the agent near the base, engaging enemies that get too close, and repairing any damage to the base as needed.
4. **Collect power-ups**: Power-ups scattered throughout the game can give the agent an advantage, such as faster movement or stronger attacks. A goal could be set for the agent to collect as many power-ups as possible to increase its chances of survival and success.
5. **Complete levels**: Ultimately, the AI agent's goal would be to complete as many levels of the game as possible. This would require a combination of survival skills, strategic thinking, and effective use of weapons and power-ups.

* **Environment**
  + 1. **Map layout:** The layout of the map where the game takes place, including walls, obstacles, and power-ups.
    2. **Enemies**: The number and type of enemies that the agent must face, including their behavior and attack patterns.
    3. **Terrain**: The different types of terrain that the agent must navigate through, such as water, ice, or sand.
    4. **Power-ups**: The different power-ups that can spawn on the map, including speed boost, invincibility, and increased firepower.
    5. **Time**: The time limit for each level and the remaining time the agent must complete its objectives.

**Phase 2 Documentation**

**Battle City with AI Agent - Project Documentation**

**Introduction**

"Battle City with AI Agent" is a classic video game where a player controls a tank and fights against enemy tanks to protect their base. In this project, we implemented an AI agent that can play the game and defeat the enemy tanks.

**Technologies Used**

This project was implemented on Jupyter notebook using Python programming language. The following libraries and modules were used in this project:

* NumPy: For numerical computations and array operations
* OpenCV: For image processing and computer vision tasks
* Pygame: For game development and event handling

**Game Environment**

The game environment was developed using Pygame library. The game environment consists of a game screen, player tank, enemy tanks, bricks, and steel walls. The player tank and enemy tanks can move in four directions (up, down, left, right) and can shoot bullets to destroy bricks and enemy tanks. The player tank can move and shoot bullets using arrow keys and spacebar, respectively.

**AI Agent**

The AI agent was developed using a Q-learning algorithm. Q-learning is a reinforcement learning algorithm that learns an optimal policy for an agent in a Markov decision process (MDP) environment. In this project, the Q-learning algorithm was used to train the AI agent to learn an optimal policy for playing the game.

The AI agent observes the game environment using image processing techniques and extracts the relevant information such as the position of player tank, enemy tanks, bricks, and steel walls. The AI agent then uses this information to make a decision on which action to take (move up, down, left, right, or shoot bullet).

**Training**

The AI agent was trained using the Q-learning algorithm. The Q-learning algorithm updates the Q-values for each state-action pair based on the rewards obtained by the agent. The rewards in this project are defined as follows:

* +10 for destroying an enemy tank
* -10 for getting destroyed by an enemy tank
* -1 for every time step taken

The Q-learning algorithm was run for a fixed number of episodes, and the Q-values were updated after each episode. The Q-values were saved to a file after training, so that they can be used for testing the agent.

**Testing**

The AI agent was tested on the game environment using the Q-values obtained from training. The agent plays the game using the optimal policy learned during training. The agent's performance was evaluated based on the number of enemy tanks destroyed and the number of times the agent got destroyed by enemy tanks.

**Conclusion**

"Battle City with AI Agent" is a project that demonstrates the use of reinforcement learning algorithms to train an AI agent to play a video game. The Q-learning algorithm was used to train the AI agent to learn an optimal policy for playing the game. The AI agent was able to successfully play the game and defeat the enemy tanks. This project can be extended to include more complex game environments and more advanced reinforcement learning algorithms.

**References**

1. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
2. Pygame documentation: <https://www.pygame.org/docs/>
3. OpenCV documentation: <https://docs.opencv.org/4.5.2/index.html>
4. NumPy documentation: <https://numpy.org/doc/stable/>