AutoPark Project Report

Group Information:

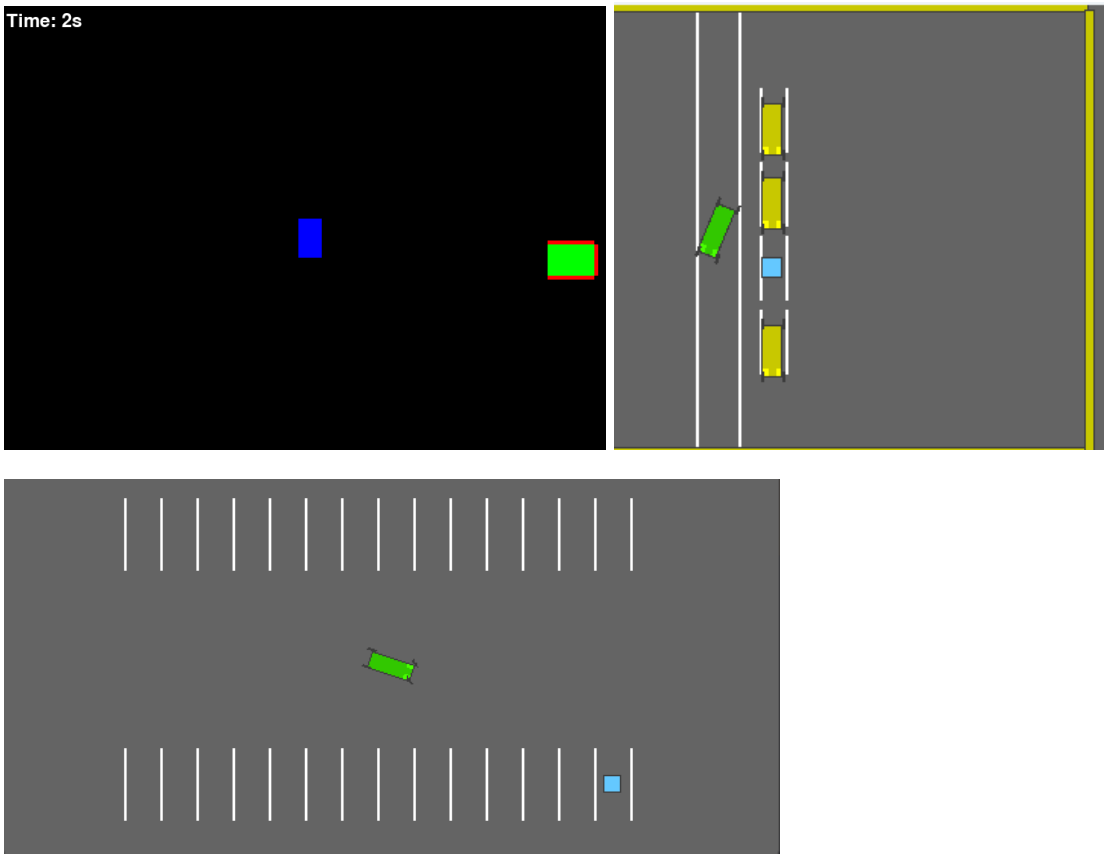
Group Name: AutoPark Innovators
Group Number (Canvas): Group 28
Members:

Andy Liu - 301472847 - abl5@sfu.ca
Mark Yun - 301474593 - mya115@sfu.ca
Jake Bareng - 301456427 - jba181@sfu.ca

System Explanation:

Our system uses different reinforcement learning techniques and algorithms to train an AI agent to perform an autonomous parking task. The environments we tested the agents on are a mix of OpenAI gymnasium environments and a custom-built PyGame environment. The parking task involved both tighter parallel parking and open lot parking. In the parking lot scenario, a desired parking position was randomly generated at each iteration, rewarding the agent if they drove into it while aligning with the parking space. Rewards were deducted if the agent was too far. The same applies to the other parking scenarios.

Environments:



AI Pipeline:

Data Input:

- Observation Space: [x, y, vx, vy, cos_h, sin_h]
- Goal state: Parking spot position and orientation
- Reward signal: Weighted by our custom scales

Data Preprocessing:

- Observations are scaled by [100, 100, 5, 5, 1, 1] to standardize ranges
    - I.e., divide positions by 100

Model:

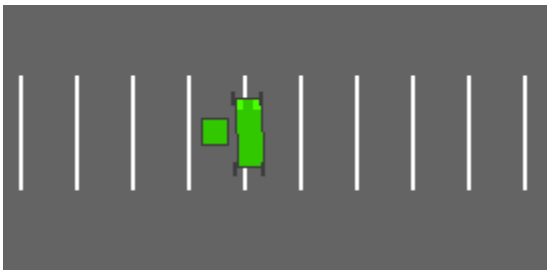
- We used various models for each environment:
    - SAC
    - DDPG with HER
    - Model-Based
    - PPO

- DQN

Output:

- Model outputs [throttle, steering] actions to control the car and direct it towards the parking space
- If model parks correct, the output is is_success = true

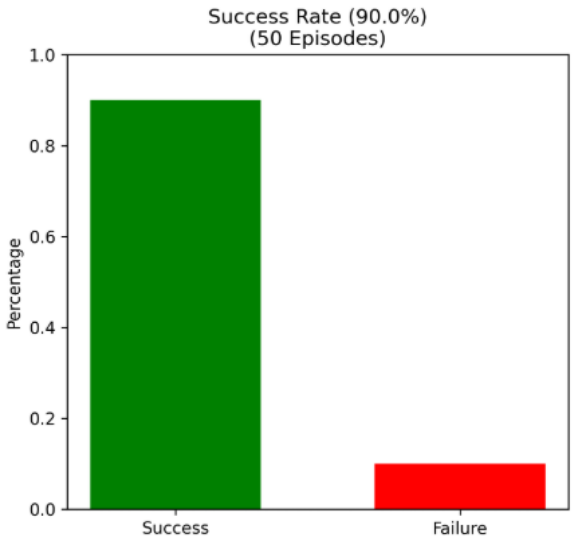
Limitations:

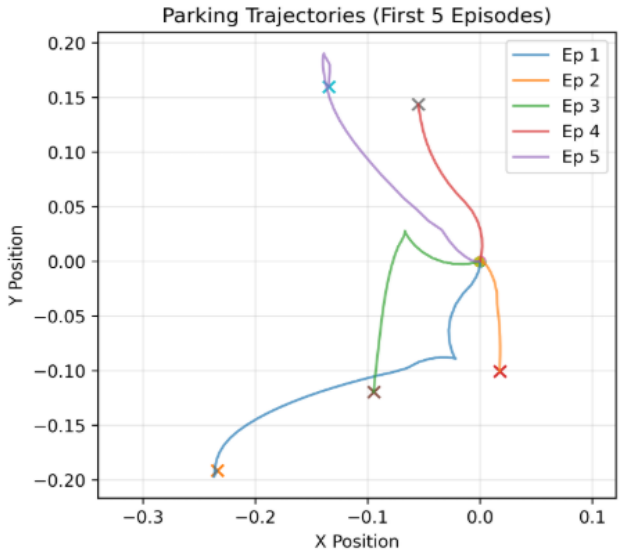
- The model-based agent had poor performance during training, which cause training to take a very long time
  - Mitigated by enabling GPU usage
- Model-based agents don't train or learn properly.
  - The system requires more time and research done to fix issues
  - Due to time constraints, we were unable to fix it and correctly implement it
- The PPO algorithm struggles with proper parking alignment, and it often parks off-centre.
  - 
  - Tuning of reward weights were done to mitigate this issue
    - However, there are still a few cases of it occurring
  - Further tuning of reward weights for the x position of the car relative to other features (y, vx, vy, steering angle) is necessary to completely remove it
  - Tuning of reward scaling for the x-axis could also help
- Parallel parking environment still needs improvement in terms of reward for speed

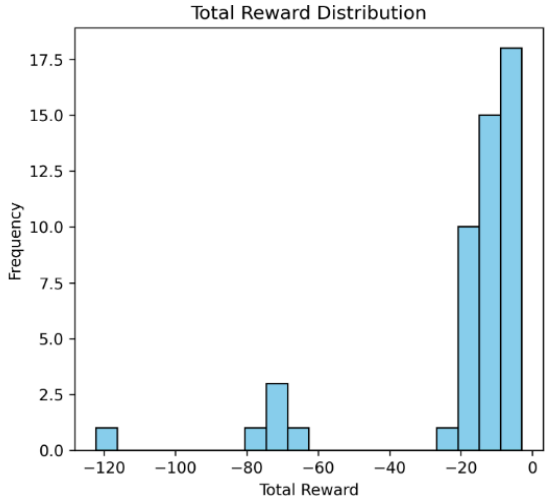
AI Methods:

Parking Lot Scenario:

- PPO
    - We chose PPO as one of the algorithms because it works well with environments that have continuous action spaces.
    - PPO's clipping prevents it from overshooting favourable policies. This is preferred in our scenario where small adjustments are important
    - The algorithm is on-policy, which works well for this experiment because it continually makes adjustments to the same policy. This keeps it stable.
    - This algorithm was implemented using the stable_baselines3 library with a vectorized environment for better training efficiency.
        - Also has two separate networks for the policy and value, which are both large ([512, 512, 512]) and therefore complex enough to handle parking. These were required for reaching comparative accuracy with DDPG and HER.
        - A simple ReLU activation function is used for better training results.
        - Trained with 300000 timesteps
    - Data and Results:
        - Training was much slower than DDPG with HER
        - The agent had issues with parking alignment and often got stuck, as seen in the video
        - Reward distribution was spread out, indicating less consistency
        - PPO Testing Video

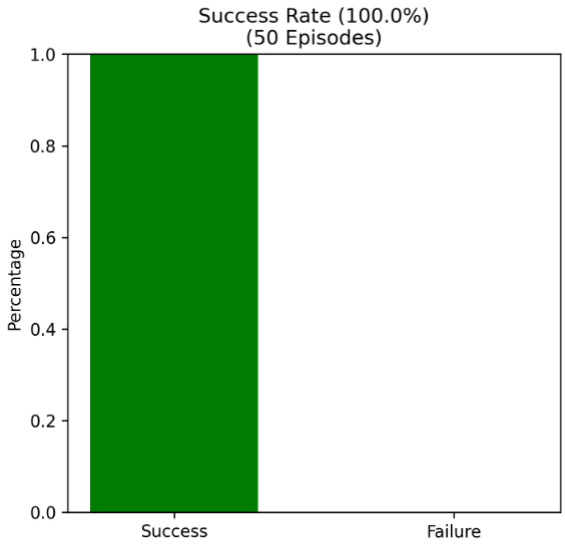


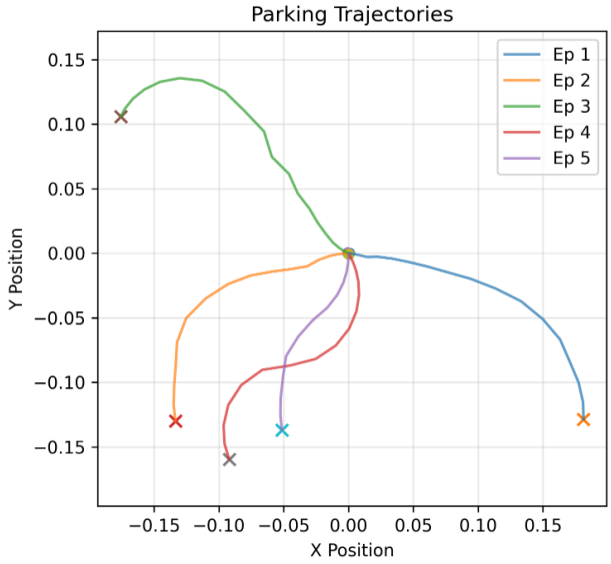


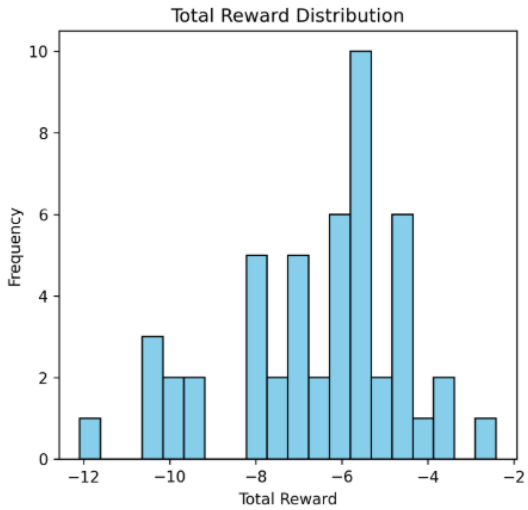


- DDPG-HER
    - DDPG, as an off-policy algorithm, was used to compare on-policy and off-policy training differences. DDPG is also commonly used for continuous action spaces that match our environment.
    - HER also making training extremely efficient and allows the agent to learn very fast
    - The stable_baselines3 library is used for implementation, and similar hyperparameters to PPO are used.
        - Trained with 80000 timesteps compared to 300000 from PPO

- Data and Results:
  - DDPG with HER performs extremely well for this task
  - Trains fast due to the HER but is also very accurate
  - It has a higher accuracy compared to PPO as well as a tighter reward distribution.
  - Overall, this algorithm is desired over the PPO for training an autonomous parking agent based on the results.
  - [DDPG-HER Testing Video]





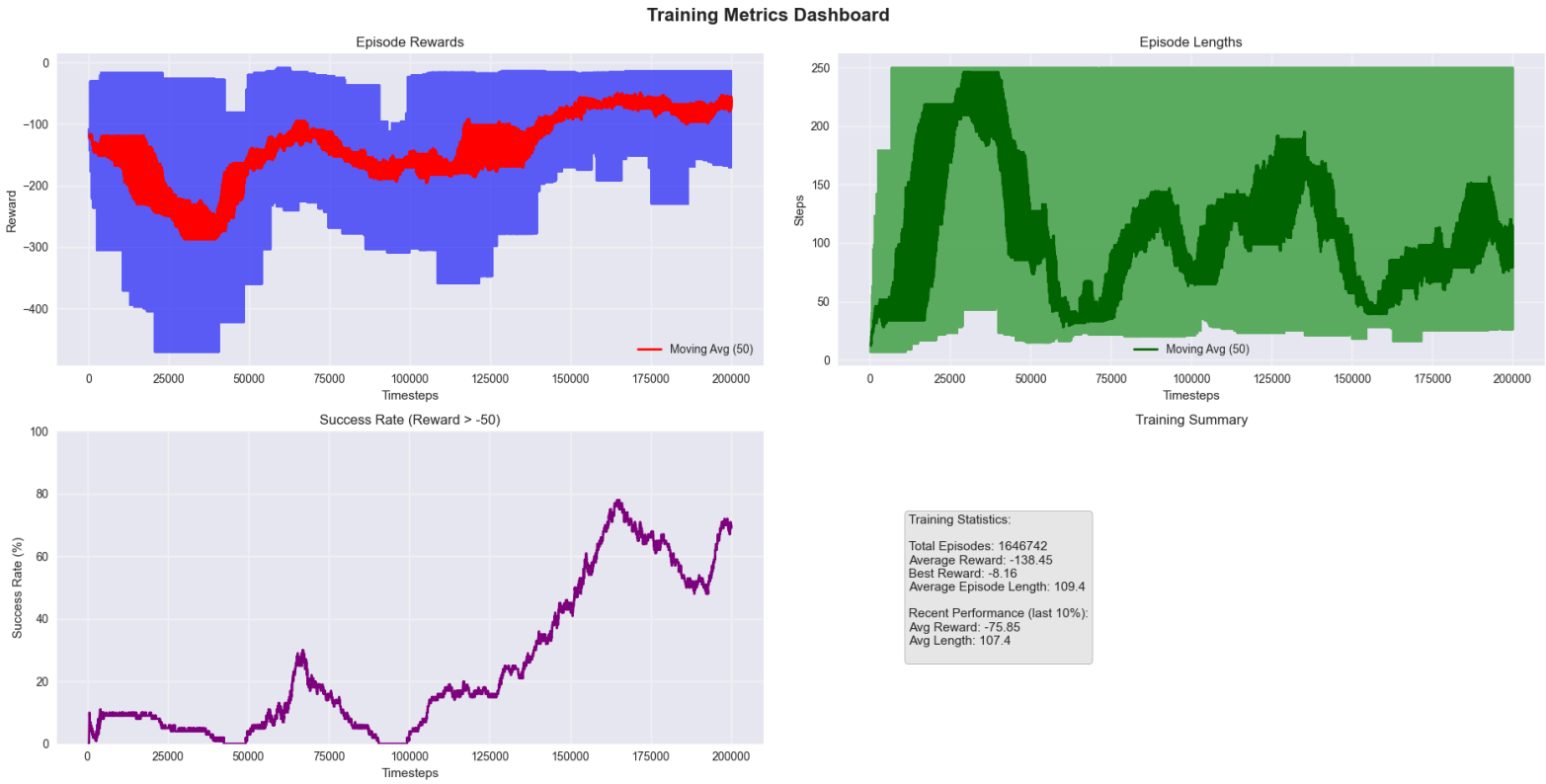


- Model-Based

    - Chose this approach because the system dynamics of the problem are relatively simple, but actual policies are difficult to learn.
        - Making a model of the environment would then be relatively simple and could help predict future actions or states.
    - It would also help to compare with the other model-free algorithms.
    - This implementation was unsuccessful and failed to learn the environment properly.
    - More time was needed to research and design model-based systems.

Parallel Parking Scenario:

- We chose SAC for the parallel parking scenario because it is well-suited for continuous action spaces.
- This algorithm was implemented using the stable-baselines3 library with Gymnasium as the interface for the environment. The networks used for training were configured with a size of [512, 256, 128], providing enough capacity to learn parallel parking.
- To encourage better generalization, the environment randomizes both the parking spot location and the starting point for each episode. This prevents the agent from simply memorizing fixed trajectories and avoids overfitting.
- The training metrics indicate that the agent shows a clear learning trend over 200,000 timesteps. Initially, the agent's performance is poor, with low rewards and a low success rate. However, as training progresses, all key metrics show significant improvement, suggesting that the agent is successfully learning the parallel parking task.
- Parallel Parking with AI: SAC Reinforcement Learning in Action

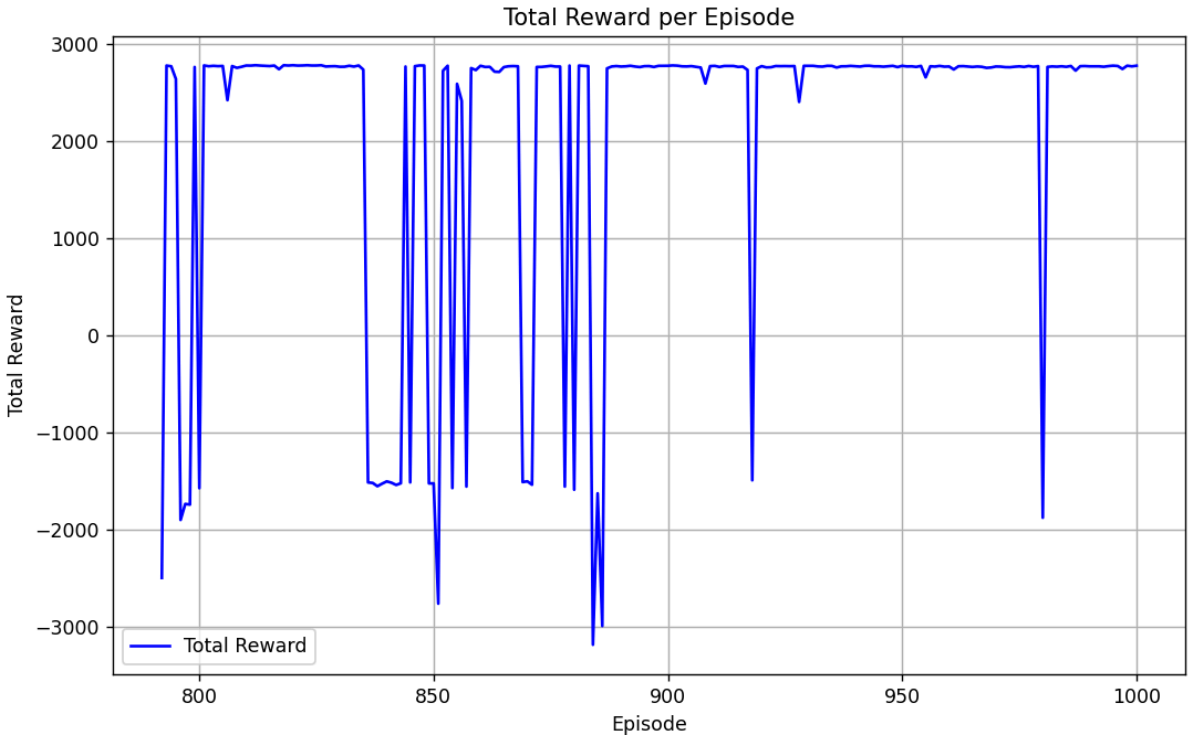


Custom Pygame Scenario:

In this part we made a fun parking game and attempted to train an AI model to play it:

- **Algorithm**: Deep Q-Network (DQN), a value-based RL method that approximates the optimal action-value function using a neural network.
- **Network Architecture**: A fully connected neural network with two hidden layers (128 neurons each, ReLU activation) and an output layer for 9 actions.
- **Training Process**:

    - The training process utilizes a replay buffer with a capacity of 10,000 to store experiences such as the state, action, reward, next state, and completed state.
    - Samples are batched (size 64) for training, minimizing MSE loss with the Adam optimizer (learning rate 0.001).
    - Employs epsilon-greedy exploration (epsilon decays from 1.0 to 0.01 with decay rate 0.995).
    - The system updates the target network every 10 episodes to ensure stability, using gamma 0.99 for discounting.
    - Trains for 1000 episodes, each limited to 600 steps (~10 seconds at 60 FPS).
- **Reward System**:
    - **Time Penalty**: -2.0 per step to encourage efficiency.
    - **Collision Penalty**: 0.15 (small, as the collisions multiply greatly when the car and the borders overlap). If the penalty is any higher, the model avoids the parking spot altogether. Adjusting for this parameter was the most difficult among all others.
    - **Success Reward**: +1000.0 plus a bonus (1.0 per remaining step) for parking successfully.
    - **Failure Penalty**: -100.0 if max steps (600) are reached without parking.
    - **Distance Penalty:**
        - We apply a negative distance penalty (-2 * the normalized distance to the spot centre).



We trained this model with 1000 episodes. This data shows the total reward per episode for around 200 episodes (episodes 800-1000.) The total rewards for each episode exhibit a polarized pattern, with values either significantly positive or significantly negative. The difference is due to the nature of the reward system and how much a car is rewarded for beating the game (and how much it is penalized for losing); this value reflects the discrepancy in total score between a failure and a success. Since this chart only shows the latest 200 episodes, I will briefly explain the model's behaviour for the first 0–500 episodes.

The agent primarily moves around the map in a random manner without any specific strategy. Increasingly, up to around episode 600, we saw that the agent's total reward starts to improve slightly, but the agent still struggles to beat the game. Between episodes 600 and 800, the rewards are mostly negative, but the total rewards occasionally jump to between

2000 and 2800, indicating that the agent knows how to beat the game once in a while. Past episode 800, the AI model starts to succeed and fail at a fairly mixed rate, as shown in the image. Past episode 900, we start to see that the agent begins to beat the game quite consistently and only fails occasionally

Feature Table:

Description	Platform	Completeness	Code	Author(s)	Notes
PyGame scenario	Local	4	Python	Mark	We reduced the scope of our game to make training shorter and easier for the model. The original game had parking spots appearing in random positions, but the new game sets the parking spot statically at the top and gives it only 6 seconds rather than 60. The change was necessary because it enabled our model to train on each episode significantly faster and improve from being unable to beat the game at all, even after 1000 episodes, to occasionally beating it after 500 episodes and becoming very consistent after 800 episodes.
(Algorithms used	Local	5	Pyth	Mark	Trained model to play game using

for Pygame)			on		Deep-Q-network
PPO Algorithm (Parking Lot Scenario)	Local	5	Python	Andy	The PPO algorithm, derived from stable_baseline3, operates in vectorized parallel environments. The algorithm performs with high accuracy but slow training.
Model-Based Algorithm (Parking Lot Scenario)	Local	2	Python	Andy	The model has issues with training, does not perform well, and requires adjustments to both hyperparameters and the overall model design.
DDPG-HER Algorithm (Parking Lot Scenario)	Local	5	Python	Andy	The DDPG algorithm, derived from stable_baseline3, utilizes the HER. The algorithm also utilizes vectorized parallel environments. The algorithm performs with very high accuracy and rapid training compared to the PPO algorithm.
Parallel Parking (SAC training)	Local	5	Python	Jake	Used SAC from stable-baselines3 due to its suitability for continuous action spaces. Displays plots of training data.
Testing (Parallel Parking SAC model)	Local	5	Python	Jake	Tests the model and displays key data like success rate, reward distribution, episode length

External Tools & Libraries:

External Python Libraries:

- Matplotlib
- Stable_baseline3
- PyTorch
- Pygame
- OpenAI Gymnasium
    - Farama Org. Highway env

References and Open Source Code:

- Reference for model-based system implementation
    - Implementation of Decision Making Algorithms in OpenAI Parking Environment | by Pradeep Gopal | Medium
    - Theory, concepts, and design were inspired by the design described in the article
    - However, more research and time was needed to correctly implement it
- Farama Parking Environment documentation: Parking - highway-env Documentation
    - Code Reference: GitHub
- Learning Algorithms:
    - PPO — Stable Baselines3 2.7.1a0 documentation
    - DDPG — Stable Baselines3 2.7.1a0 documentation
    - HER — Stable Baselines3 2.7.1a0 documentation
    - SAC — Stable Baselines3 2.7.1a0 documentation

Installation Steps:

- Required libraries
    - Gymnasium, highway_env, numpy, matplotlib.pyplot, collections, stable_baseline3, torch
    - Install commands:

```
pip install stable_baseline3
pip install torch
```

pip install numpy
pip install matplotlib
pip install gymnasium
pip install highway_env

- NOTE: Specific installations of Torch and Python are required if you wish to run these algorithms off your GPU. Please refer to the how-to guide for details. This information is not required for the programs to function.