# Project Summary

*The game is set in a 4x4 grid representing a small parking lot. At the start, there are five cars, each occupying a single 1x1 space within the grid. Additionally, certain spaces may contain b**arriers, which act as obstacles and restrict movement. Each car can move forward or backward within the lot and according to the car’s orientation but is unable to pass through other cars or blocked spaces. The objective is to strategically maneuver each car off the grid’s edges, one by one, until all cars have been successfully removed. This process requires careful planning, as freeing up space for one car may be necessary to allow others to exit.*

# Propositions

*•**C(i, j, o): This proposition is true if there is a car in the grid at position (i, j) facing orientation o at the current state. The orientation o could be:*

*•0: Facing up and down*

*•1: Facing right and left*

*•**E(i, j): True if the cell at position (i, j) in the grid is empty.*

*•M(i, j, d): This proposition is true if car at position (i,j) moves in direction d, matching its current orientation*

•*B(i, j): True if the cell at position (i, j) contains a barrier or wall that the car cannot pass through.*

*TODO:*

• *Allow expansion of the board so that it does not have to be a 4x4 grid*

• *Include a time variable in to our propositions to keep track of all past turns.*

# Constraints

*•Occupied or Empty: Each cell in the grid can either be occupied by a car or be empty.*

*•∀i,j:(C(i,j,o)∨E(i,j)) for all orientations o.*

*•Movement Constraints: A car can only move if there is no barrier or another car directly in front of it.*

*•If a car at (i, j) is facing right and is going to move, the cell (i', j) must be empty:*

*•C(i,j,o)∧M(i,j,d)→E(i',j)∧¬B(i',j).*

*•Similar constraints apply for other directions.*

*•Update Position: If a car moves, its previous position becomes empty, and its new position is occupied.*

*•If car ccc at (i, j) facing right moves, then:*

*•M(i,j,d)→(E(i,j)∧C(i',j,o)).*

*•Similar constraints for other directions.*

*•No Collision: No two cars can occupy the same cell at the same time.*

*•∀i,j:¬(C(i,j,o1)∧C(i,j,o2)).*

*•Exit Condition: Define the exit cell as (x\_exit, y\_exit). A car can only be removed from the grid if it moves into this cell.*

*•C(xexit,yexit,o)→E(xexit,yexit) when the car moves into the exit.*

*• ∀i,j:E(i,j) ensures that every cell (i,j) on the grid is empty, with no cars occupying any space.*

# Model Exploration

In the development of our parking lot game model, we explored various aspects and iterations to ensure the functionality and constraints of the game mechanics were effectively captured. Here’s a summary of the ways we have explored our model:

1. **Initial Prototyping**: We started with basic classes for Car and Barrier, implementing core functionalities such as position initialization, orientation handling, and basic movement capabilities. This helped establish the foundational structure of the grid and the vehicles within it.
2. **Constraint Definition**: We defined various propositions and constraints to represent the rules of the game. For instance:
   1. **Occupied or Empty**: Ensuring that each grid cell is either occupied by a car or empty.
   2. **Movement Constraints**: Implementing rules for how cars can move, including checks for barriers and collisions with other cars.
   3. **Update Position**: Ensuring the model accurately reflects the state of the grid after a car moves.
3. **Proposition Logic**: We explored the use of propositions like *C(i,j,o)*  for car presence and orientation, *E(i,j)* for empty cells, and *B(i,j)* for barriers. By implementing dynamic classes for these propositions, we were able to create more complex constraints using logical operations such as conjunctions and disjunctions.
4. **Future Enhancements**: We have outlined plans for future versions of the model, such as expanding the grid dynamically and incorporating a time variable to track past moves. This will enhance the model's complexity and realism.

Additionally, we created a randomized state of the game to be used each time for testing.

# Jape Proof Ideas

*List the ideas you have to build sequents & proofs that relate to your project.*

*Logical Statments*

* *If a car is facing a certain direction and it is not blocked in that direction then it is able to exit.*
* *If a tile does not contain a car or a barrier then it must be empty*
* *If a car is blocked by a barrier in both directions, then the game cannot be won*

*TODO: add more and express them as sequents to solve in Jape.*

# Requested Feedback

* *In our python implementation, we have our current board set-up as a randomly generated board each time, should we keep it like that or do it as a predetermined board as you did in the tsuro example?*
* *Are we on the right track by creating the classes that we have?*
* *To proceed any further in our python code we would like to know if what we have so far is correct and if not, what can we do to change it?*

# First-Order Extension

*To extend this model to a predicate logic setting, we would represent each of the game's entities, actions, and constraints as logical predicates:*

* *Using predicates to represent the states and conditions of each game entity.*
* *Defining movement actions with preconditions to enforce valid moves.*
* *Adding temporal elements to track moves over time.*
* *Specifying goal constraints for exiting cars and clearing the board.*
* *Including logical constraints to avoid invalid states like collisions or out-of-bounds moves.*