

**SANTA CLARA UNIVERSITY**

Department of Computer Engineering

I HEREBY RECOMMEND THAT THE THESIS PREPARED  
UNDER MY SUPERVISION BY

Carter Duncan, Jack Cunningham, Andrew Wang, Alexander Kennedy

ENTITLED

Urban Planning Optimization via “Cities: Skylines”

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF

**BACHELOR OF SCIENCE**  
IN  
**COMPUTER ENGINEERING**

---

Thesis Advisor(s)

date

Ying Liu

6/4/2021

Department Chair(s)

date

Darren Atkinson

6/4/2021

# Urban Planning Optimization via “Cities: Skylines”

By

Carter Duncan, Jack Cunningham, Andrew Wang, Alexander Kennedy

## **SENIOR DESIGN PROJECT REPORT**

Submitted to  
the Department of Computer Engineering

of

SANTA CLARA UNIVERSITY

in Partial Fulfillment of the Requirements  
for the degree of  
Bachelor of Science in Computer Engineering

Santa Clara, California

2021

# Abstract

Im Blee bla ble blu

\*aight this should be essentially a summary of the entire thesis and project in like 2-3 short and succinct paragraphs\*

\*someone should read this abstract and know everything about this project from a high level\*

\*problem, solution, implementation, etc\*

# Acknowledgements

We would like to extend a special thanks to our advisor Dr. Ying Liu for sharing her extensive knowledge with us on Artificial Intelligence and Deep Q Learning techniques. The insight we gained from meetings with her, as well as from her Artificial Intelligence course helped prepare us for this project implementation.

# Table of Contents

## SENIOR DESIGN PROJECT REPORT

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Table of Contents</b>	<b>v</b>
<b>Chapter 1: Introduction</b>	<b>1</b>
1.2 Problem Statement	1
1.2 Background and Existing Research	1
1.2.1 Machine Learning Background	1
1.2.2 Reinforcement Learning Background	2
1.2.3 Alpha Go	2
1.2.4 Open AI Five	2
1.3 Proposed System	3
<b>Chapter 2 : Requirements</b>	<b>3</b>
2.1 Functional	3
2.1.1 Powerful computing platform to run game	3
2.1.2 Computing platform to run Tensorflow	3
2.1.3 Extract relevant information from game	3
2.1.4 Train our agent using game data and python	3
2.2 Non-functional	4
2.2.1 Extraordinary amounts of computing power	4
2.2.2 A UI for quick parameter access	4
2.3 Design Constraints	4
2.3.1 Training environment not completely accurate	4
2.3.2 Lack of Computer power	4
2.3.3 Limited Training Time	4
<b>Chapter 3: System Architecture</b>	<b>5</b>
3.1 Cities:Skylines C# API	5
3.1.1 Modding API	5
3.1.2 Using the modding API for AI	5
3.1.3 Game Data Capture	5
3.1.4 Perform Game Actions	6
3.2 Python AI agent	6
3.2.1 Game Data Capture	6
3.2.2 Game State Analysis	6
3.2.3 State-to-Goal Calculation	6

3.2.4 Action Selection	6
<b>Chapter 4: Technologies</b>	<b>7</b>
4.1 Hardware	7
4.2 Software	7
4.2.1 Steam	7
4.2.2 Cities: Skylines	7
4.2.3 Visual Studio	7
4.2.4 Github	8
4.3 TensorFlow	8
<b>Chapter 5: Design Rationale</b>	<b>8</b>
5.1 Hardware	8
5.2 Software	8
5.2.1 Steam	8
5.2.2 Cities: Skylines	8
5.2.3 Visual Studio and Github	9
5.2.4 Python TensorFlow	9
<b>Chapter 6: Testing</b>	<b>9</b>
6.1 Test Plan	9
6.2 Testing Process	9
6.3 Testing Results	10
6.4 Testing Limitations	11
<b>Chapter 7: Research Applications</b>	<b>11</b>
7.1 Reinforcement Learning	11
7.2 Urban Planning	11
<b>Chapter 8: Future Research</b>	<b>12</b>
8.1 Quantum Simulations	12
8.2 Crowd sourced machine learning	12
<b>Chapter 9: Societal Issues</b>	<b>12</b>
9.1 Ethical	12
9.2 Political	13
9.3 Health and Safety	13
9.4 Environmental Impact	13
9.5 Compassion and Empathy	14
<b>Chapter 10: Conclusion</b>	<b>14</b>
10.1 Summary	14
10.2 Lessons Learned	14
10.3 Advantages and Disadvantages	14

10.4 Future Work	15
10.4.1 Training agents on more complex maps	15
10.4.2 Adding agent actions and features	15
10.4.3 Distributed agent training	15
<b>Works Consulted</b>	<b>16</b>
Urban Planning	16
Reinforcement Learning	16
Cities: Skylines	16
<b>Appendix</b>	<b>17</b>
Cities: Skylines Mod Documentation	17
Compiling the Cities Skylines Mod	17
Preparing the Cities: Skylines Game Environment	18
Launching the Mod and the Agent	18
Troubleshooting	19
Cities Skylines Mod Public Classes and Functions	19

# Chapter 1: Introduction

## 1.2 Problem Statement

The problem observed in the world around us is the poor planning of urban development and layouts. Many urban layouts are impractical, ineffective, and overall poorly implemented. Reactions to this may include building bigger and better buildings. However, this often overlooks the problem at hand, which is, often, not the problematic buildings but the configuration and layout of the city itself. Metropolitan areas across the United States, and the world, suffer from these inefficiencies. This results in a large market and thus makes a solution imperative.

## 1.2 Background and Existing Research

The team conversed with a few urban planners and designers from varying firms in different cities such as New York City and Boston. These planners granted much-needed insight into the existing and primary method undertaken for urban planning. Usually, a group of human designers, architects, and consultants gather around a physically constructed model of the city or portion considered for construction or modification. While this is useful for bringing multiple different skill sets and perspectives to one place to discuss, it is a rigid and limited experience. This method has difficulties surrounding reflecting and representing the sheer amount of metrics necessary to understand the entire city properly. Often multiple iterations are required to reach a final solution. Iterating, in this case, involves modifying multiple physical models several times, which is time-consuming. In urban projects, the quicker the iterations, the sooner the group can converge on an idea. Our team is hoping to provide a solution that will reduce the convergence time. Quicker idea convergence will save all stakeholders lucrative time, money, and resources throughout this process.

Currently, some other teams exist that are attempting to resolve similar problems. The primary one in the market is Sidewalk Labs. Sidewalk Labs is an urban innovation company that creates products to tackle city problems. While they have several products that range from increasing parking space or reducing home energy, they have one product that offers similar approaches to urban planning. Delve is a tool they created to allow designers and developers to visualize different layouts of neighborhoods and associated metrics and an overarching score, much like our agent does. Sidewalk Labs is a sophisticated contender in the market segment. Not many other significant contenders have entered this sphere, meaning not enough solutions exist for this glaring problem, and thus it will take more time until most urban landscapes become efficiently designed and implemented.

### 1.2.1 Machine Learning Background

Machine Learning is a subset of computer science and artificial intelligence that allows computers to train on large data sets to recognize patterns to make informed decisions on new data. These computer algorithms improve automatically through experience based on the use of this data. Three primary forms of machine learning exist, Imitation, Supervised, and



Reinforcement. Imitation Learning is when an agent ‘follows’ along with a human, much like a master-apprentice relationship, hoping to ‘imitate’ their process. Fields such as modeling and animation utilize Imitation Learning. Supervised Learning is when an agent trains on a labeled data set. Then the agent is given data that is not labeled and draws on its previous knowledge, the trained dataset, to make predictions on the new unknown dataset. Fields such as image recognition commonly use Supervised Learning. Reinforcement Learning, used in this research project, requires a framework and system for the agent to understand how to operate. Often in Reinforcement Learning (RL) projects, no “correct” output or result is known. The agent possesses a reward system that guides the agent through multiple states to determine which actions to take. RL results in the agent independently learning its patterns and “playstyle.”

Machine Learning (ML) entered the realm of video games relatively recently, where it is often using training models combining Imitation Learning and Reinforcement Learning. Certain video games that have seen the application of RL so far include Chess, GO, and Dota2. Currently, ML has only been applied to video games to see its feasibility. Minimal research shows how ML training can develop and utilize applicable real-world skills beyond these video game environments.

### **1.2.2 Reinforcement Learning Background**

Reinforcement Learning (RL) is a subset of Machine Learning (ML), where an agent possesses the tools and framework to solve a problem and finds its patterns and solution implementation. Functionally, an agent has an environment (referred to as a state) to observe. For each state, the agent has a set of possible actions to execute. Each action will result in a new environment (or state). A reward value is associated with each action-state pairings, with a transition function implementing the movement from one state to the next. A policy then dictates how the agent should choose amongst the varying action-state pairs based on the reward values.

RL is a heavy iteration process. RL is a learn-by-doing sort of implementation. The agent is treated almost like a newborn baby, with no previous knowledge of the system, no bias, and no experience. The agent dives into the deep end of the pool of learning. The agent must make decisions and struggle through the consequences of each. Through this struggle, it gains insight and experience in the system. Thus it will learn a policy-reward system that provides an updated, expected reward value to each action-state pair. As it trains, it gains a better understanding of which actions prove beneficial and which actions prove detrimental. Thus the agent may learn patterns and strategies that human users would have failed to identify or understand. Astute pattern recognition can prove immensely useful if applied to a system that can benefit from unseen optimizations, such as the urban planning sector.

### **1.2.3 Alpha Go**

AlphaGo is a program created to play Go developed by an AI research company acquired by Google. The program used a Monte Carlo search algorithm combined with artificial neural networks. The program was the first of its kind, managing to beat several professional Go players, a feat previously impossible for computers.

### **1.2.4 Open AI Five**

Open AI Five is a program developed to play the game Dota 2. This system utilized reinforcement techniques to train its program by playing over 10,000 years of games against

itself. Similar to Alpha Go, Open AI Five was the first computer system of its kind to beat professional players.

## **1.3 Proposed System**

Our proposed system is a similar system to the previous systems. Our system is an AI agent that will use reinforcement learning techniques to learn and train in the environment of the game Cities: Skylines. This system will be able to run the game Cities: Skylines and make actions in order to “play the game” and design a city. The reinforcement learning aspect of the agent will allow it to improve and eventually create optimized cities.

# **Chapter 2 : Requirements**

## **2.1 Functional**

### **2.1.1 Powerful computing platform to run game**

- CPU: Intel Core 2 Duo, 3.0GHz or AMD Athlon 64 X2 6400+, 3.2GHz.
- CPU SPEED: Info.
- RAM: 4 GB.
- OS: Microsoft Windows XP/Vista/7/8/8.1 (64-bit)
- VIDEO CARD: nVIDIA GeForce GTX 260, 512 MB or ATI Radeon HD 5670, 512 MB (Does not support Intel Integrated Graphics Cards)

### **2.1.2 Computing platform to run Tensorflow**

- Python 3.6–3.8. Python 3.8 support requires TensorFlow 2.2 or later.
- pip 19.0 or later (requires manylinux2010 support)
- Ubuntu 16.04 or later (64-bit)
- macOS 10.12.6 (Sierra) or later (64-bit) (no GPU support) ...
- Windows 7 or later (64-bit) ...
- GPU support requires a CUDA®-enabled card (Ubuntu and Windows)

### **2.1.3 Extract relevant information from game**

A script written in C# was necessary to extract variables from the game environment such as population count and pollution levels. C# is required as it is the language Unity uses, the game engine used to create Cities: Skylines

### **2.1.4 Train our agent using game data and python**

A operating system capable of utilizing pipes is required to move data between the C# script previously discussed and the Python program using TensorFlow to train our agent. The agent will adjust itself based on the game data until it reaches a win criteria or maximum allowed iteration. While the program will function without large amounts of computing power is required to see meaningful results.

### **2.1.5 Custom Game Mod**

A mod, short for modification, will have to be build using the specifications provided by Cities: Skylines to allow for our custom C# code to be injected into the game.

## **2.2 Non-functional**

### **2.2.1 Extraordinary amounts of computing power**

While not absolutely necessary, many previous projects which have been successful with the reinforcement learning technique we applied here have had access to millions of dollars which could be spent freely on cloud computing credits to power these computationally intensive processes. To have a final product able to be used in the design and creation of real cities I estimate around 10 million dollars would be required.

### **2.2.2 A UI for quick parameter access**

A simple web based UI would allow for anyone even if they didn't have programming knowledge to modify the critical system parameters to easily make changes and see what works and what doesn't

## **2.3 Design Constraints**

### **2.3.1 Training environment not completely accurate**

As it is not possible to effectively use the real world as a training environment for the game we settled for the next best thing in the form of the game Cities: Skylines. While the game does its best and is surprisingly complex, it is still way away from accurately modeling how people, buildings, and so much more interact with each other in ways that would be impossible for a computer game to model.

### **2.3.2 Lack of Computer power**

While this has been previously touched on it is important to reiterate that the computing power we had access to was limited to our personal desktop computers. These computers are very powerful and all contain GPUs but don't come close to projects such as Open AI Five which had thousands of times more computing power available to them.

### **2.3.3 Limited Training Time**

The time constraints of being a project required to be completed by graduation was already constricting and compounded the problem that training an artificial intelligence also takes a long time and can only be started once all the other components are fully functional. These facts resulted in a very limited training time for our agent.

# Chapter 3: System Architecture

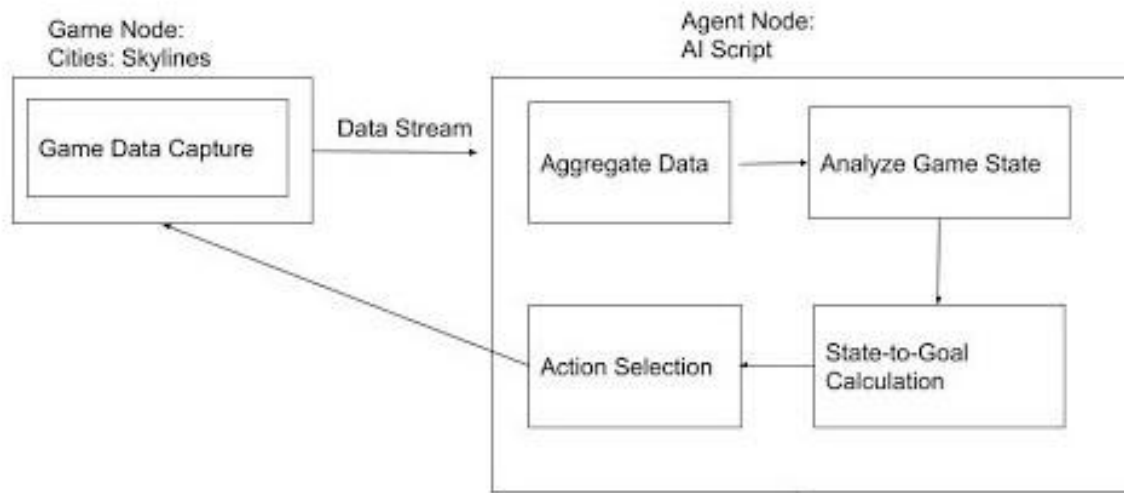


Figure 1: Architecture of the implemented system

## 3.1 Cities:Skylines C# API

### 3.1.1 Modding API

The C# aspect of the system is built for the modding API of Cities: Skylines. Cities: Skylines is a game built with the game engine Unity, and supports publicly created game modifications to be created and shared with the game's online community. The game allows for compiled C# files that fit the Unity C# code format to be run alongside the game when placed in a "mod" directory, and when explicitly enabled in the settings menu of the game. When enabled, active mods will have their "start()" function executed immediately after the load time of a level, and have their "update()" function executed every 1ms during game time. Because these mod files are executed alongside the game, they are free to declare objects that are exclusive to the game files, and use the public functions and variables of any aspect of the game code, as if they were written and compiled in the same workspace as the game code.

### 3.1.2 Using the modding API for AI

The modding API allows the agent to interact with Cities: Skylines directly, which eliminates the need for either player inputted actions or mouse and keyboard macros to be used to execute the actions of the agent's choosing. Because the modding API allows for a new C# file to execute game functions and retrieve game variables, the C# aspect of the system has two major tasks within the larger system: capture game data, and perform game actions.

### 3.1.3 Game Data Capture

The C# aspect of the agent is responsible for obtaining the performance metrics from the game state. The game stores and updates variables relevant to a city's success, and a C# mod running alongside the game is able to retrieve these variables and use them to determine how well the city is doing. The agent utilizes 28 performance metrics which are obtained in this manner.

### **3.1.4 Perform Game Actions**

The C# aspect of the agent also must execute actions in the game environment as requested by the AI. Because the C# mod is integrated with the game code when enabled, the agent is able to execute actions that would normally be executed by a person with a mouse and keyboard. These actions include placing buildings and zoning areas, which are the two critical aspects of city creation within this game environment. Thanks to the modding API, the agent is able to create buildings and zones in similar ways to a player performing the same action with a keyboard and mouse.

## **3.2 Python AI agent**

### **3.2.1 Game Data Capture**

The first step is to acquire data the agent will learn from. This information, such as population count, income, and available power, is aggregated using the previously discussed C# script and sent to the python application through a data pipe and saved in Python objects.

### **3.2.2 Game State Analysis**

Using the game state information acquired in the previous step, the AI agent will have to make a decision about which action to take. At the start of training the agent's actions are seemingly random. As the agent begins taking actions, each action will result in a new game state with each metric being affected positively or negatively. As the agent makes these choices it learns the benefits and drawbacks of the resulting game state from each action. This results in the agent slowly realizing that certain actions will produce more desirable game states, reflected via higher scoring results.

### **3.2.3 State-to-Goal Calculation**

After an action is taken by the agent it will be given a reward score depending on how well it aligned with a set of criteria the user defined. As we chose to use a Deep-Q Learning network architecture, for each possible action the agent can take, the agent calculates an expected reward for each action and resulting state pairing. Then the agent chooses the action with the highest reward. If the user wants to ensure the city has an ample supply of power and the agent took actions that resulted in the power supply being overtaken by power demand, the reward score given to the agent would be lower.

### **3.2.4 Action Selection**

Using the scores discussed above, the agent will choose the action with the highest score. This action will then be encoded and sent over a system pipe to the C# script where the action will be decoded and the corresponding action will be taken in the Cities: Skylines game. For example, one possible selected action could be to build a hospital in square (2,7). This information would be encoded as an integer such as "96024" and sent over as discussed.

# Chapter 4: Technologies

## 4.1 Hardware

The computer that ran Cities: Skylines, the mod, and the artificial intelligence agent had the following hardware components:

Graphics card: GeForce RTX 2070 SUPER

CPU: Intel(R) Core(TM) i7-9700KF CPU @ 3.60GHz

RAM: 32 GB

This type of setup is generally regarded as high performance for a personal computer and would likely be labelled as a high-end gaming PC. However, these labels are only true for singular personal computers. In terms of comparing our total computing power to other artificial intelligence training projects, our computing power significantly falls behind the projects of the last few years.

## 4.2 Software

### 4.2.1 Steam

Steam is a platform for online PC games and software distribution. We use Steam to access Cities: Skylines as well as the community content available for Cities: Skylines in the Steam Workshop. Steam is required for running our agent, as we make use of multiple mods from the Steam Workshop, such as the “Remove Power Lines” and “Remove Pipes” mods.

### 4.2.2 Cities: Skylines

Cities: Skylines is a city management game developed by Colossal Order and published by Paradox Interactive. We found that this game offers a workable balance between simplicity and features that made it suitable for our project. The general gameplay loop for Cities Skylines is as follows:

- Build roads

- Assign zones adjacent to roads

- Build infrastructure and specific service buildings to serve the nearby zones

- Adjust budget and policies to meet needs

- Purchase additional land to expand into

The game can be paused and run at 1x, 2x, and 3x speed. At 1x speed, the game does not run anywhere near reality’s time scale, as in-game days pass in real-time seconds. Cities: Skylines is not city simulation software, but a city management video game, so many differences exist between the video game state and reality for the sake of having a better entertainment experience.

### 4.2.3 Visual Studio

We used Microsoft Visual Studio to work on the C# side of the project. Visual Studio provided increased efficiency with coding, debugging, compiling, and testing. Visual Studio allowed us to load dependencies and create post-build scripts to make compiling and testing code automatic. In addition to this, we used a Github plugin to enable Visual Studio to commit, push, and pull files to the online Github repository.

#### **4.2.4 Github**

Github is an online version control service. It implements a repository model to allow multiple contributors to update their local files with changes from the repository and to update the online repository with local changes. It allowed our team to easily collaborate asynchronously on software such as the python and C# scripts.

### **4.3 TensorFlow**

Tensorflow is a free and open source software library written by Google that provides all the machine learning functionality we used during our project. This saved us enormous amounts of time as we didn't have to write our machine learning code from scratch.

## **Chapter 5: Design Rationale**

### **5.1 Hardware**

At the beginning of the senior design project, we had planned on using our allocated budget on purchasing a high-performance personal computer to aid in running the agent. With a budget of 1,500 USD, we planned out a build that would focus on computing power. However, we never made use of this build nor budget, as there were several issues with the plan. Firstly, each of our members were remote, so we would have to choose only one member to send the parts to and to assemble the computer. In addition to this issue, the graphics card market was in its worst state for purchasing, with prices being significantly higher than a year before. Because of these logistical problems, we did not build a dedicated machine to run the agent, and instead made use of what we had. While not every member had a high-performance personal computer, we had at least one that was suitable enough for running the agent as a proof of concept.

### **5.2 Software**

#### **5.2.1 Steam**

We chose to use Steam for a couple of reasons. Firstly, most available modding information and tutorials used the Steam version of Cities: Skylines. Secondly, the Steam Community Workshop allowed us access to useful user-created mods. We used three mods during development: Remove Pipes, Remove Power Lines, and Loading Screen Mod. The Loading Screen Mod is useful for monitoring the details of what is being loaded during the loading period and the amount of memory that is being used, but it is not required to run the mod. The Remove Pipes and Remove Power Lines mods are required, as we used them to reduce the complexity of the game environment, also decreasing the amount of actions that the agent would need to take. We made this decision to allow us to bring the agent to the “minimum viable product” state within the limited time we had, and we found that building pipes and power lines were lower priority compared to creating buildings and zones.

#### **5.2.2 Cities: Skylines**

At the start of the senior design project, we were certain that we wanted to work with artificial intelligence. As a recent example of a successful artificial intelligence project, we looked up to the OpenAI Five project that created an agent to play the video game DotA 2. With this example in mind, we wanted to create an artificial intelligence agent that could play a video game. After

brainstorming, we decided upon Cities: Skylines to be that game. At that point in time, none of the members had extensive experience playing the game themselves, but we had some knowledge of the mechanics of the game through watching playthroughs and livestreams. One of the reasons that we chose this game in particular was because of the visibility of results. When watching experienced players create cities, the end products were interesting to look at in themselves. The ways in which roads and buildings were placed to make the most of available land by players created a sort of artwork resembling a circuit board. We wanted to find out what an AI designed city would look like.

### **5.2.3 Visual Studio and Github**

These applications and services were used to aid development. Neither the game nor the agent strictly depend upon these applications and services. While writing the C# code and Python code, we used them to increase our efficiency. Without them, it would be unlikely that the agent would have reached a proof of concept or minimum viable product stage by the end of the time allotted to us.

### **5.2.4 Python TensorFlow**

While it is possible to build the entire artificial intelligence agent within the C# mod, this would mean creating the reinforcement learning functions from scratch. Instead of making everything from the ground up, we utilized Python's abundance of machine learning and artificial intelligence packages which are available to the public. We decided to take advantage of these packages by using TensorFlow to handle the reinforcement learning part of the agent, while leaving only the data gathering and action implementation on the C# side. All of the decision making would be done on the Python side. This posed a new challenge of how to communicate between the C# side and Python side of the agent, which we solved with named pipes.

## **Chapter 6: Testing**

### **6.1 Test Plan**

Because we face hardware and time-related limitations in obtaining a completed agent to test the functionality and results of, our ideal outcome of the testing phase was to prove that our system could follow the training process on a much smaller scale and fulfill our described requirements in section 2. This involved running a significantly smaller amount of iterations during the training process of our agent to determine if it meets our requirements or not. The outcome of this plan would determine only the functionality of our system, and would not yield a well trained agent, as a trained agent would require far more data points than we had the processing power and time to obtain with iterations.

### **6.2 Testing Process**

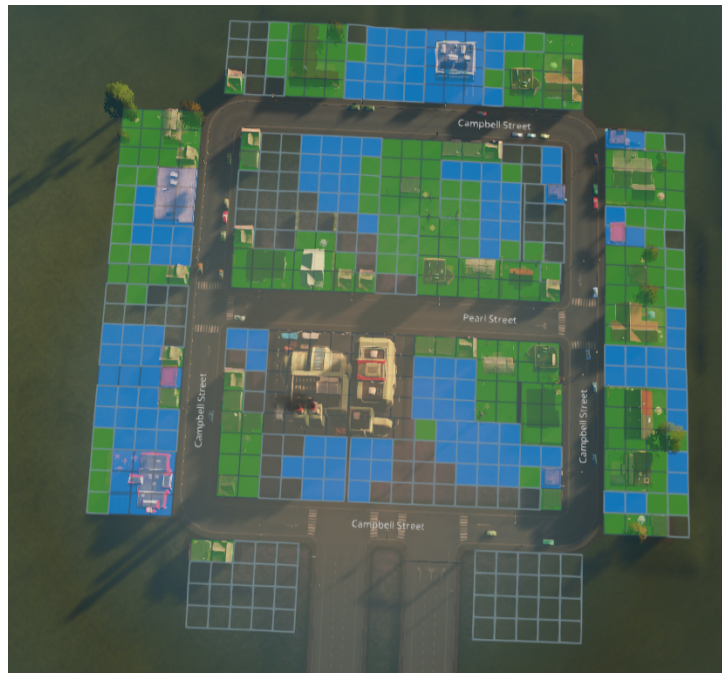
Testing our completed prototype involved setting up our agent with greatly reduced data gathering time by decreasing the amount of collection steps of both the training environment and the evaluation environment. To test the requirements of our system, we ran our agent on a standard action count of 100 actions per step at a rate of one action per 7 seconds (a step being one runthrough of the game on our test map), but only collected 5 training steps and 5 evaluation



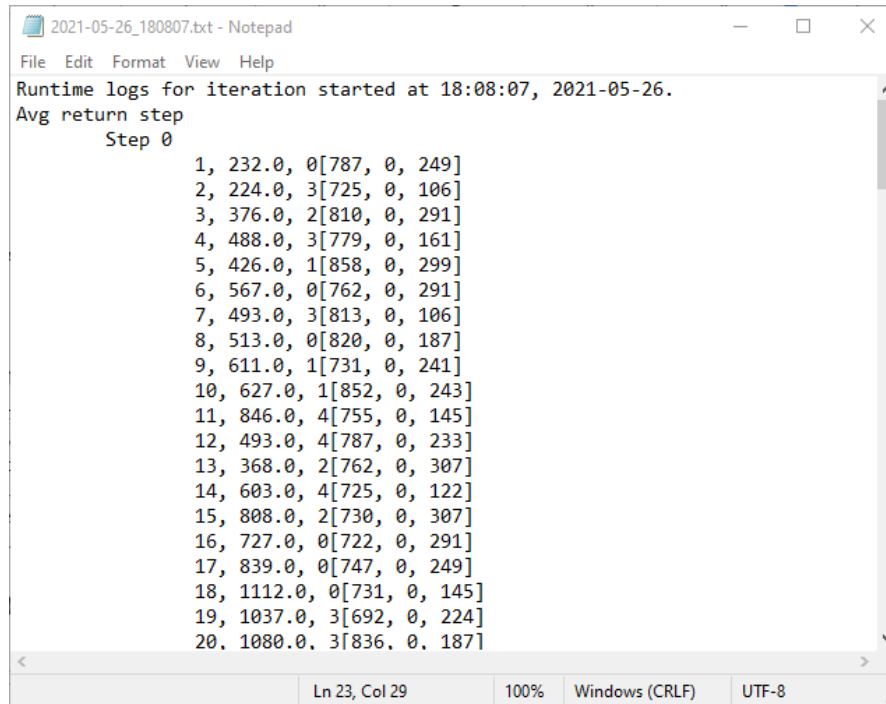
steps. This allows us to determine if our prototype is functioning within our expectations, but does not yield a trained agent.

## 6.3 Testing Results

In total across all testing we were able to run 50 iterations of our game environment across multiple sessions. Our system was able to successfully complete the training process while meeting the requirements that we were testing for. The system was able to successfully link its C# and python components, load into the game, read game data, take actions based on the agent's decisions, and iterate actions every 7 seconds until the end of a step. After one step our agent was able to process the data obtained and adjust itself, automatically reload the start of the level, and start another step. As intended, the limiting factor of our system was the game time, in that actions need to be spaced out across time in order to see the effect that each action has on the game. Our system displayed scores for each step taken, which fulfilled the requirements of intuitive results, and did not crash frequently.



*Figure 2: One completed city constructed by the agent, with zones color coded*



```
2021-05-26_180807.txt - Notepad
File Edit Format View Help
Runtime logs for iteration started at 18:08:07, 2021-05-26.
Avg return step
Step 0
1, 232.0, 0[787, 0, 249]
2, 224.0, 3[725, 0, 106]
3, 376.0, 2[810, 0, 291]
4, 488.0, 3[779, 0, 161]
5, 426.0, 1[858, 0, 299]
6, 567.0, 0[762, 0, 291]
7, 493.0, 3[813, 0, 106]
8, 513.0, 0[820, 0, 187]
9, 611.0, 1[731, 0, 241]
10, 627.0, 1[852, 0, 243]
11, 846.0, 4[755, 0, 145]
12, 493.0, 4[787, 0, 233]
13, 368.0, 2[762, 0, 307]
14, 603.0, 4[725, 0, 122]
15, 808.0, 2[730, 0, 307]
16, 727.0, 0[722, 0, 291]
17, 839.0, 0[747, 0, 249]
18, 1112.0, 0[731, 0, 145]
19, 1037.0, 3[692, 0, 224]
20, 1080.0, 3[836, 0, 187]
```

Figure 3: A section of a log file opened in Notepad

## 6.4 Testing Limitations

This testing process was only able to test the functionality of the agent, not its ability to repeatedly produce well designed cities. To test this, our system would need to run for much longer and in parallel in order to obtain enough data points to fully train the reinforcement learning agent. While we could not test our agent once it is fully trained, by meeting the requirements it can be inferred that our system would be able to fully train an agent if implemented on a larger scale.

# Chapter 7: Research Applications

## 7.1 Reinforcement Learning

Any future group of similar size can review our progress and see the techniques we utilized and some common pitfalls. To our knowledge, this is the first project to ever apply reinforcement learning to a city-building game such as Cities: Skylines. As the field is so complex it would have been highly beneficial to have a project like this to look at while creating ours, but that was not the case. However, anyone seeking to follow in our footsteps now has a guidebook to explore this uncharted territory.

## 7.2 Urban Planning

Due to the current scope of this project, it seems well suited to improve current research in Urban Planning. Due to the nature of city development it is too time consuming to do real time research as cities take years to build and a small error in planning could result in dramatic costs. Having a virtual environment to test ideas alongside the AI agent could prove to be very effective as rapid iteration would be possible.

# Chapter 8: Future Research

## 8.1 Quantum Simulations

Quantum computers will supposedly perform accurate simulations of our natural world at a capacity more significant than any traditional computer. When this technology is better developed and available to the public, it could simulate a realistic model of a city, and a computer agent could be trained based on this realistic model for much more accurate and informative results.

## 8.2 Crowd sourced machine learning

One of the most significant barriers to small teams trying to work on reinforcement learning is the cost of acquiring the computing power necessary. The existence of a platform that would allow people to volunteer a portion of their computers processing power to projects they thought had potential would allow research in this field to progress at a significantly faster rate.

# Chapter 9: Societal Issues

## 9.1 Ethical

This project involves modifying urban landscapes, where multitudes of people reside. Ethical issues are abundant, making the identification of and response to these issues more lucrative. Our team analyzed this project from multiple perspectives to ensure we could address as many issues as possible.

The most critical ethical quandary is the prospect of gentrification. Gentrification refers to when a poor area of a city "experiences an influx of middle-class or wealthy people who renovate and rebuild homes and business," and this usually results in "an increase of property values and the displacement of earlier, usually poorer residents"

(<https://www.merriam-webster.com/dictionary/gentrification>). The manifestation of this in our project is that given an existing neighborhood layout, the agent, naturally, may attempt to replace low-income neighborhoods with high-income neighborhoods. Boosting area income is objectively understandable as higher valued neighborhoods bring more residents and more income. However, this is not always desired in the real world as not all residents possess the same skill sets and financial situations to support this lifestyle. Allowing this modification to occur would result in a large displacement of residents, inciting much unrest in the city. Additionally, making a neighborhood more valued and expensive does not guarantee an influx of residents that match that criteria. Too many high-value areas are not a guaranteed solution and often glosses over the reality of the problem that has resulted in these low-income neighborhoods in the first place.

Another issue is the lack of 1:1 parallels between the game world and the real-world environments. This simulated virtual game environment is not fully representative of the real world. Real-world urban landscapes can not be fully represented and encapsulated within this particular digital virtual environment. Thus, as is seen many times throughout this paper, the conclusions drawn from the game agent may not always be fully applicable and relevant. Human analysis and validation of all game agent results are crucial and imperative. The developed game

agent is a sophisticated tool to be wielded correctly. Great care and consideration must be present to leverage the agent's full potential. The city residents should remain at the forefront of all urban design.

Expanding further on the above paragraphs, our agent cannot fully factor in the human element of cities. While the agent can make decisions that affect metrics representative of people, such as happiness, education, and more, human elements in this environment get abstracted away to a certain extent. Human citizens themselves are not variables in this environment or for the agent. Additionally, actual residents are far more complex than modeled simple in-game populations. Demographics do not exist here. The addition of demographics would introduce extensive levels of complexity, and potentially, elements of demographic bias into the resulting city. For example, historically, minorities such as African Americans and Latinos/Hispanics are usually in lower-income neighborhoods, while Caucasians are generally in higher-income neighborhoods. If the agent possessed demographic information such as this, it might recreate segregated neighborhoods correlating to income levels. Segregation is not ideal. Biased data leads to biased results.

## **9.2 Political**

Human beings are governors and the governed; thus, political considerations are factored into urban reworkings, such as district planning and redrawing. Unfortunately, there may be cases where users discard specific results from our project as they may infringe upon certain political realms. Our agent does not interject itself within political frames of ‘thinking’ for cities. Minimal elements such as policies can be implemented further down the road, but they are limited in their scope and impact.

## **9.3 Health and Safety**

Emergency Medical Services is a crucial component of any urban layout. Ample coverage for fire-prone areas is needed to prevent colossal property damage and the loss of lives. Police are required to assist with urban activities and prevent crime, which detriment businesses and individual lives throughout the city. Health and medical services are a must for individuals of all ages. These services are individual buildings in the game environment, providing a ‘zone of coverage’ for surrounding buildings. As a result, ‘optimal’ locations can be found for these service buildings, maximizing coverage. However, a potential concern is if the agent develops a form of totalitarian approach where so long as ‘most’ people have coverage, it is complacent with certain areas not having coverage. Ideally, all citizens want access to life-saving services. Great care is needed to ensure certain minorities retain access to these services.

## **9.4 Environmental Impact**

Urban areas often negatively influence environmental metrics like ‘greenery’ as cities remove trees and plants to replace them with roads and buildings. Additionally, with large volumes of people living close, even with advanced plumbing and sanitation, waste and litter are bound to accumulate. While our agent can recommend areas to provide things such as plumbing, the actual implementation is in the hands of the city planners. Our agent can suggest areas to renovate into parks, recycling centers, and greenhouses, but how to do that is beyond this project’s scope. Whether parks use soil-friendly fertilizers are decisions that other third parties must weigh.

## 9.5 Compassion and Empathy

An essential element to reiterate throughout this entire paper and this process are that this agent is computer code. Even if it developed human-like intelligence, it would not possess the many dynamic and intricate elements of humanity that make us so complex. Emotions, compassion, and empathy are unable to be represented and encapsulated in our agent.

# Chapter 10: Conclusion

## 10.1 Summary

Machine learning and artificial intelligence are brilliant fields that can produce technical tools that lend themselves to discovering and assisting the problem-solving process. However, not all problems are created equal, and neither are machine learning models. A precise model and implementation may be the best solution for each problem one encounters. When discussing what type of artificial intelligence agent we wanted to create to play Cities: Skylines, we considered reinforcement learning, unsupervised learning, and supervised learning. We found that reinforcement learning best suited our current objective. As a result, we learned about what goes into a reinforcement learning agent by creating and testing each part, ranging from gathering data on the game-state to calculating the reward function. By the end of the design process, all of our members have gained insight into the details of reinforcement learning and gained insight into the software engineering process.

## 10.2 Lessons Learned

One of the more specific lessons we learned was the balance between processing power and training time for reinforcement learning. In our case, we found that we had relatively little of both, which led to the agent not being fully trained or trained to a high standard of play. This lesson can be generalized beyond reinforcement learning and applies to balancing time and resources when undergoing projects.

Along with all the large-scope, general lessons that we learned, there were also many small things, like using existing applications, services, and packages to speed up our workflow. We learned how to set up a project timeline and execute it in reality and adjust it to meet unexpected changes. We learned how to present information on a complex topic and communicate important information to audiences not specialized in artificial intelligence or computer science. The senior design process was an experience of creating something new and an experience of constantly learning and growing.

## 10.3 Advantages and Disadvantages

As seen earlier, our project's processing power fell short compared to other reinforcement learning projects in the last few years. The sheer processing power that similar research projects use is staggering, far greater than anything a group of 4 college students could muster, even with the option of university funding. Unfortunately, we do not have the capital to acquire this computing power and likely will not anytime soon.

In addition to lack of computing power, another challenge was the sheer time required to effectively and adequately train this agent. At this point, the agent is still in the very early stages

of learning and is nowhere near taking the optimal actions to reach or progress towards a goal. Reinforcement learning requires a significant amount of time to produce valuable results.

Reinforcement learning also has many different reward policies, optimization methods, and many other parameters to tweak, change and optimize. We tinkered around with many as we attempted to understand the impact and significance of each better. With the limited amount of time, we could not fully explore all the possibilities that this brought.

## **10.4 Future Work**

### **10.4.1 Training agents on more complex maps**

The training map we used was designed primarily for testing purposes and is not representative of an actual city. The testing map we designed was on flat land with no water features or other geographical features. While this might represent a few cities, it is far from ideal as many big cities have a vast array of complex features. In the future, the agent would train on maps created with more complex designs so it would be able to handle these challenges whenever it encountered them.

### **10.4.2 Adding agent actions and features**

We decided to prioritize creating buildings and assigning zones for the agent's actions during the development process. In doing so, we left infrastructure such as roads, pipes, and power lines out of the mix. In addition to this, we also left out some game features such as buying new land and creating budgets and policies. For future work with this agent, adding these actions and features to the agent will undoubtedly increase complexity and significantly increase the agent's ability.

### **10.4.3 Distributed agent training**

Eventually, we will need the computing power that more than one single machine can provide. The solution to this challenge is to distribute the computing load across a vast array of less powerful computers and combine the results from each. Usually a complicated task, but thankfully Tensorflow provides built-in distribution functionality to be done with minimal configuration.

# Works Consulted

## Urban Planning

<https://www.merriam-webster.com/dictionary/gentrification>

<https://www.re-thinkingthefuture.com/architects-lounge/a1517-10-examples-of-bad-urban-city-planning/>

## Reinforcement Learning

<https://towardsdatascience.com/how-to-teach-an-ai-to-play-games-deep-reinforcement-learning-28f9b920440a>

<https://ieeexplore.ieee.org/document/8022767>

<https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/>

<https://www.geeksforgeeks.org/what-is-reinforcement-learning/>

<https://deepmind.com/research/case-studies/alphago-the-story-so-far>

<https://cdn.openai.com/dota-2.pdf>

[Cloud Tensor Processing Units \(TPUs\) | Google Cloudhttps://cloud.google.com › Cloud TPU › Documentation](https://cloud.google.com/tpu/docs/tpu-overview)

## Cities: Skylines

[https://skylines.paradoxwikis.com/Modding\\_API](https://skylines.paradoxwikis.com/Modding_API)

<https://community.simtropolis.com/forums/topic/67088-index-of-modding-tutorials-information/>

<https://community.simtropolis.com/forums/topic/73404-modding-tutorial-0-your-first-mod/>

# Appendix

## Cities: Skylines Mod Documentation

### Compiling the Cities Skylines Mod

While the Cities: Skylines game has its own compiler for C# code, we used the Visual Studio method to manually compile and move the resulting file into the appropriate file location.

To replicate this, we followed the instructions for method 2 in

<https://community.simtropolis.com/forums/topic/73404-modding-tutorial-0-your-first-mod/>

The following instructions are from the aforementioned forum post by “boformer”

#### Set up the project

- 1) Install Visual Studio Community with the .NET desktop development feature, with .NET Framework 3.5 development tools
- 2) Create a new project with Select File → New → Project
- 3) In the templates section, select Templates → Visual C# → Windows Classic Desktop
- 4) Select “Class Library (.NET Framework)”
- 5) Select “.NET Framework 3.5” in the dropdown menu
- 6) Enter the solution name and local storage location
- 7) Set up the dependencies
- 8) Install Cities: Skylines through Steam
- 9) In the solution explorer, click “references” then “add reference”
- 10) Use the browse button and navigate to the folder  
Steam\steamapps\common\Cities\_Skylines\Cities\_data\Managed
- 11) Select the following files:
  - Assembly-CSharp.dll
  - ColossalManaged.dll
  - ICities.dll
  - UnityEngine.dll
- 12) Add the files
- 13) Set up the post build script
- 14) Right click the project in the solution explorer and select Properties
- 15) Select build events
- 16) Copy and paste the following script:

```
mkdir "%LOCALAPPDATA%\Colossal Order\Cities_Skylines\Addons\Mods\$(SolutionName)"
del "%LOCALAPPDATA%\Colossal
Order\Cities_Skylines\Addons\Mods\$(SolutionName)\$(TargetFileName)"
xcopy /y "$(TargetPath)" "%LOCALAPPDATA%\Colossal
Order\Cities_Skylines\Addons\Mods\$(SolutionName)"
```

With Visual Studio now set up with the dependencies and post build script, the C# code for the mod can now be compiled directly into the game files by clicking “rebuild solution” in Visual Studio under the “build” tab.



## **Preparing the Cities: Skylines Game Environment**

The mod and agent development so far has operated on a limited version of the Cities: Skylines mod. Some complexities of the game environment have been removed in order for us to be able to create a minimum viable product given a limited time period and computing power. The limitations are within the game environment and the actions that the agent can take. In order for the agent to run properly, the game environment must have the following mods installed.

### Remove Power Lines

To simplify the game environment, we decided to remove the infrastructure requirement of having power lines connect electricity generators to electricity consumers. In order to change the game environment to this, we used a publically available Cities: Skylines mod available through the steam workshop. Go to following link:

<https://steamcommunity.com/sharedfiles/filedetails/?id=572888650>

and click the “subscribe” button. The mod should download automatically. Open Cities: Skylines, in the main menu, click the content manager. Select the mods tab on the left, and make sure the relevant mods are enabled.

### Remove Pipes

We also decided to remove the infrastructure requirement of having pipes connect water and sewage systems. Repeat the mod installation steps detailed in the Remove Power Lines section, but with the link: <https://steamcommunity.com/sharedfiles/filedetails/?id=576997275>

### Install the Map

Download the TEST\_MAP\_1.crp file

Move the file into C:/Users/[your user]/AppData/Local/ColossalOrder/Cities\_Skylines/Saves

If there is no such directory, launch the Cities: Skylines game, quit the game, then check again.

### Install Python Dependencies

After installing the latest version of Python 3, use pip to install the following modules:

```
pip install imageio
pip install tensorflow
pip install tf_agents
```

## **Launching the Mod and the Agent**

### Launching the Agent

First, the python script must be run before the Cities: Skylines game.

Navigate to the directory containing the file

“cities\_skylines\_deepq\_learning\_for\_final\_santa\_clara\_engineering\_project.py”

Open a windows console the same directory

Enter the following command:

```
python “cities_skylines_deepq_learning_for_final_santa_clara_engineering_project.py”
```

The agent should now be launched and awaiting the Cities: Skylines side

### Launching the Mod

Launch Steam

Launch Cities: Skylines through the Steam library

Click the load option, and select the test map

After a delay of approximately 30 seconds, observable changes should occur in-game

It is recommended to select the zoning tool in-game to observe zones being assigned

## **Troubleshooting**

### Cities: Skylines Crashes After 30 Seconds

There is a known issue with the reset and quickload functionality based on hardware. If this error is occurring on your machine, navigate to the configuration file, “skylinesconfig.txt”, that should be in the same directory as the “cities\_skylines\_deepq\_learning\_for\_final\_santa\_clara\_engineering\_project.py” python script. Set the variable “resetting\_allowed” equal to “False” in order to disable this optional feature.

## **Cities Skylines Mod Public Classes and Functions**

### Performance Measures

The performance measures class contains only public variables where various gamestate values are stored.

*void get\_performance\_measures()*

This function calls all the relevant API functions and stores the return values in the public member variables.

*void print\_performance\_measures()*

This function creates a single string with all public member variables and labels them. The string is printed to the in-game developer console (F7 to view).

*string performance\_measures\_cs()*

This function creates a single string with all public member variables separated by commas. The string is returned.

### Action Parser

The action parser class contains functions for calling the corresponding API functions based on an input string. The public member variables are meant for debugging and easy format changes for delimiter characters, which are the characters separating tokens. The token\_delimiter char is set to the symbol separating tokens in an input string. The vector\_delimiter char is set to the symbol separating values in a vector. When debug is set to true, related prints are sent to the in-game developer console.

*void parse\_actions(string)*

This function takes the input string as a parameter. It then calls the function split\_string\_to\_tokens and passes it the input string as well as a delimiter symbol, defined as a member variable. split\_string\_to\_tokens returns a list of strings, which is then passed to the function run\_tokens. The string should be formatted with the name of the action as the first token, followed by the values of the parameters. The parameters must be able to be parsed from string representation to the expected value type. Each token should be separated by the token\_delimiter character (currently set to '|'). In the case of a vector being passed as a parameter, each value should be separated by the vector\_delimiter char (currently set to ',').

*list<string> split\_string\_to\_tokens(string, char)*

This function takes an input string and delimiter character as parameters. The string will be split into substrings between each delimiter character. The substrings are then stored in a list of strings which is returned after all tokens have been read.

*vector3 string\_to\_vector3(string)*

This function takes an input string as a parameter. The string will be split into substrings, similarly to the `split_string_to_tokens` function, but in this case, the substrings will be converted to floats. The corresponding x, y, and z values are formatted into a `vector3` and returned.

*void run\_tokens(List)*

This function takes a list of strings as a parameter. The first token will be compared in a switch statement to set actions. Currently, the possible first token values are either the string “createbuilding” or the string “createzone”. If the first token does not match either, no action will occur. Once the corresponding switch body has been entered, the expected parameters will be parsed from the second token onwards. If the token’s string value is incompatible with the appropriate value type, the game may crash or the mod may restart due to the exception. Once the arguments are set to the appropriate types, the API function will be called with the arguments, executing the action in-game.

## DataReader

The `DataReader` class is the core of the mod, and extends `MonoBehavior`, which allows Unity to run the class alongside the rest of the game code during runtime. This class contains private functions `Start()` and `Update()` which are executed by Unity’s mod interface automatically while *Cities: Skylines* is running. The `DataReader` class contains the main action classes which allow selected actions to be executed in game, as well as the threaded function in charge of sending and receiving data from the python agent, and a function to reload the game from the most recent save in order to “reset” the agent between episodes.

*bool CreateBuilding(out ushort, uint, Vector3, float, int)*

This function takes 5 arguments. The first is a `ushort` passed as an out variable, which will be set to the ID of the building when it is constructed, or 0 if it was unable to construct the building. The remaining 4 are the building’s prefab ID, position, angle, and hitbox length. This function fetches the prefab, checks for collision errors, then places the specified building using a public function of the class that handles construction within *Cities: Skylines*. The return value is true if the construction was successful, or false if there were collision errors.

*int ZoneArea(Vector3, int, quantity)*

This function takes 3 arguments: The position of the zoning as a `Vector3`, the type of zone to be placed as an `int`, and the quantity of zones to be zoned in the area. This function determines the closest zonable tile to the specified location and zones it with the specified zone using a public function of the class that handles zoning with *Cities: Skylines*. This process is repeated until the specified quantity of zones have been zoned, or when no more zones are found within 5000 pixels of the specified location. The return value is the quantity of zones that were zoned.

*void sendData()*

This function starts the thread which reads and writes to the named pipes that communicate with the Python element of the system. This thread is only created once and operates through resets of the game.

*void reset()*

This function loads the game to the most recently saved save file using a public function of the class that handles loading within Cities: Skylines. This allows the agent to reset the state of the game to the beginning between iterations.