Simulating Real-World Human Activities with VirtualCity: A Large-Scale Embodied Environment for 2D, 3D, and Language-Driven Tasks

by

Jordan Ren

S.B., Computer Science and Engineering,
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored By: Jordan Ren
Department of Electrical Engineering and Computer Science
May 12, 2023

Certified by: Antonio Torralba
Professor
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Simulating Real-World Human Activities with VirtualCity: A Large-Scale Embodied Environment for 2D, 3D, and Language-Driven Tasks

by

Jordan Ren

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Embodied environments act as a tool that enables various control tasks to be learned. Within these simulators, having realistic rendering and physics ensures that the sim2real gap for tasks isn't too large. Current embodied environments focus mainly on small-scale or low-level tasks, without the capability to learn large-scale diverse tasks, and often lack the realism for a small sim2real gap. To address the shortcomings of current simulators, we propose VirtualCity, a large-scale embodied environment that enables the learning of high-level planning tasks with photo-realistic rendering and realistic physics. To interact with VirtualCity, we provide a user-friendly Python API that allows the modification, control, and observation of the environment and its agents within. Building this realistic environment brings us closer to adapting models trained in simulation to solve real-world tasks.

Thesis Supervisor: Antonio Torralba
Title: Professor

Acknowledgments

I would like to thank Antonio Torralba for his guidance and support throughout my time in his lab. Thanks to his knowledged insight we have been able to make large strides in our work on VirtualCity. I would also like to thank Shuang Li for her continued assistance and mentorship throughout my master program. Throughout my term Shuang has helped me work through the code base and provided valued input into improvements that could be made. Lastly, I would like to thank Kabir Swain for his great work on the development of VirtualCity.

Contents

List of Figures

Chapter 1

Introduction

1.1 Challenges

The prevalence of autonomous agents in the real world is growing. These agents can be seen navigating complex traffic patterns on the road, managing storage facilities, and helping humans with day-to-day tasks. In order for these agents to succeed in the real world, extensive testing must be done. However, doing so with physical agents and collecting real-world data is often expensive and time-consuming. This leads us to the use of simulators that aim to replicate the real world as closely as possible, such as VirtualHome[39] and Alfred[41]. Through these simulators, researchers are able to collect data for a variety of tasks, from vision to decision-making tasks. However, when transferring models from simulation-to-reality (sim2real), there can be gaps in behavior. As such, it is important that simulators being used to solve real-world tasks are as realistic as possible.

Existing embodied environments mostly focus on small-scale or low-level control tasks, such as robotic manipulation, or indoor activities, such as VirtualHome and ALFRED. Through environments focusing on low-level control tasks, researchers are able to build robots that are able to perform dexterous and complex actions. Similarly, researchers are able to study human behavior through environments that enable high-level tasks, and train agents to perform day-to-day human activities. By combining these efforts on all fronts, eventually we will be able to create a real-world agent

capable of human activities. However, currently there is no large-scale embodied environment that can cover a wide range of human activities. To study broader human social interaction and complex planning, a larger environment is needed. Furthermore, existing embodied environments lack the realism to close the sim2real gap. The current visual rendering of scenes, physics, agents, tasks that agents can complete, and objects is not as realistic as it could be.

1.2 Approach

Towards creating an embodied agent that can perform high-level human activities, we built VirtualCity, the largest multi-agent embodied environment to simulate various human social activities and decision-making tasks that are made in daily life. The VirtualCity environment will be large in scale, including indoor and outdoor environments, 2400+ indoor objects, 4500+ outdoor objects, and a multitude of tasks to complete. It uses Unreal Engine to achieve photo-realistic rendering and realistic physics, as well as multi-modal observations of the environment. Through a user-friendly application program interface (API) users can control multiple agents to test a variety of high-level control tasks, obtain observations of the environment, and modify the environment.

Within the VirtualCity environment, we enable 2D, 3D, high-level planning, and language tasks. Using the photo-realistic environment, 2D tasks such as image segmentation and detection can be learned. Similarly, realistic rendering enables 3D tasks such as 3D object generation through multiple camera views. The realistic environment should allow for high-quality inferences of the environment's state from agent viewpoints, enabling agents to potentially learn information about their environments from images in the agent's views. The Python API allows offline reinforcement learning (RL) training for high-level planning tasks at a large-scale, spanning both indoor and outdoor environments. Furthermore, language models for breaking down high-level goals can be tested within the environment.

1.3 Contributions

In this thesis, we aim to take advantage of improvements in recent releases of game engines in order to build upon the work that was done within VirtualHome, and extend the simulation to outdoor environments as well. Specifically, we aim to make improvements from VirtualHome on two fronts: realism and scale.

In terms of realism, Unreal Engine [32] is known for its advanced graphics capabilities. It has a robust lighting and shading system, realistic particle effects, and high-quality textures. The engine also supports real-time ray tracing, which can significantly enhance the realism of a scene. Another major improvement comes from Unreal Engine's physics engine. Unreal Engine has a robust physics simulation system that is based on Nvidia's PhysX [29] engine. This physics simulation system enables developers to create realistic physical interactions in their games or simulations, such as gravity, collisions, and forces. These improvements that Unreal Engine has over Unity should allow for greater realism in simulation within VirtualCity, reducing the sim-to-real gap and enabling visual based learning from agents.

A major point within the VirtualCity simulator contains both indoor and outdoor environments. As simulating large outdoor environments can be computationally expensive, we build VirtualCity using Unreal Engine, which is performant when handling large and complex scenes. It also has support for multi-threading and can take advantage of modern hardware, such as multi-core CPUs and GPUs. This is helpful when it comes to online training with agents. By enabling a larger environment in which agents can navigate, more complex and long-term tasks can be learned. Instead of confining an agent to a single location, the agent can learn tasks that span multiple environments, much as humans do. With VirtualCity, we aim to create a highly realistic environment that enables the training of autonomous agents through simulation. Furthermore, with the use of procedural generation, VirtualCity includes 100,000 unique indoor apartment environments, and can eventually be expanded to generate thousands of other indoor areas. This should enable agents trained using VirtualCity to become robust when operating in new environments.

Chapter 2

Related Works

2.1 Simulation

Through 3D simulated environments, simulators have enabled research in the fields of reinforcement learning, planning tasks, language tasks, and 2D/3D tasks. Recent works related to simulated environments typically simulate small indoor areas that enable the learning of small-scale navigation tasks [39, 42], or focus on robotic agents [36]. One such simulator is VirtualHome[39], a predecessor of sorts to VirtualCity, which allows for the creation of large activity video datasets as well as testing small-scale high-level actions in an indoor setting. Our work focuses on creating a large-scale, performant simulation, that would enable the testing of large-scale, high-level action sequences. Recently, MineDojo [33] used the popular game Minecraft as the basis for training an embodied agent that was able to perform open-ended tasks without manual labeling. While in MineDojo the agent is able to perform tasks to an outstanding ability within the game, we look towards creating a more realistic simulation environment to train embodied agents so they may be functional in the real world. Furthermore, with advances in procedural generation, ProcTHOR [31] was able to generate 10,000 unique houses to train robots to become more robust. We plan on incorporating the use of procedural generation within VirtualCity as well.

2.2 Benchmarks

Embodied AI datasets generated from simulated environments are used for a variety of tasks, such as decision-making, segmentation, and 3D modeling. Notably, BEHAVIOR-1K [42] has created a simulation that enables 1000 activities in 50 realistic and diverse scenes, resulting in high-quality datasets. Generally, more realistic environments result in better data, which environments from VirtualHome [39], BEHAVIOR-1K [42], and ALFRED [41] have improved upon with their near photo-realistic environments. However, with the release of Unreal Engine 5's (UE5) realistic physics engine and rendering, we argue that the VirtualCity environment built on UE5 brings us even closer to the real world than previous simulators. We hope to improve upon the advances made by previous simulators by building an environment that allows for numerous indoor and outdoor activities using realistic physics and rendering, allowing agents trained using VirtualCity to become more human-like. Recently, Facebook released the largest 2D dataset for segmentation to date, SAM[34], and Objaverse[30] released an annotated 3D object dataset with over 800,000 objects. As the scale and quality of SAM and Objaverse can be used for applications that would overlap with any 2D or 3D datasets generated from VirtualCity, we have opted to move our focus away from dataset building. However, the high-quality visuals and realistic physics within VirtualCity should still enable vision-based information gathering and learning for embodied agents.

2.3 Learning human activity

The representation of actions as programs has increased lately. Agents within Watch and Help (WAH) [40], Generative Agents [38], and Minedojo [33], have demonstrated the ability to complete tasks in a simulated real-world environment much as a human would. In WAH, embodied agents were trained on sequences of actions performed by a hierarchal planner, which was also used as a baseline for performance (more details in Chapter 4), which enabled them to assist other agents in completing tasks. The

idea was to infer another person's intentions and beliefs from observation. We are also interested in this concept, but on a larger scale. In a larger environment with more complex tasks, we are interested to see how agents would perform when navigating larger goals and with potentially more agents. When humans perform tasks, they often do so in the most optimal fashion. By observing many iterations of large-scale tasks, we hope that embodied agents within VirtualCity will be able to collaborate with other agents much as in WAH.

Chapter 3

Environment

3.1 Overview

The VirtualCity simulator will include both indoor and outdoor environments. The outdoor environment includes a city built from [8, 9, 6, 7, 5], with a functioning traffic system and humanoid system agents. There are 14 different humanoid system agents that can be used in city crowds or as staff in stores, each with various action states such as walking, talking, and carrying items[7]. The system agents also have the ability to perform basic tasks within the city to make the environment more realistic. There are 12 different functioning vehicles that can be driven by agents or setup to drive around the city following an open-source traffic system[9]. The indoor environments will eventually include basic indoor city environments such as convenience stores, offices, restaurants, etc. Within both environments, we include the following visual modalities:

1. RGB images
2. Segmentation maps for separate objects and classes
3. Optical flow
4. Depth images
5. Object bounding boxes

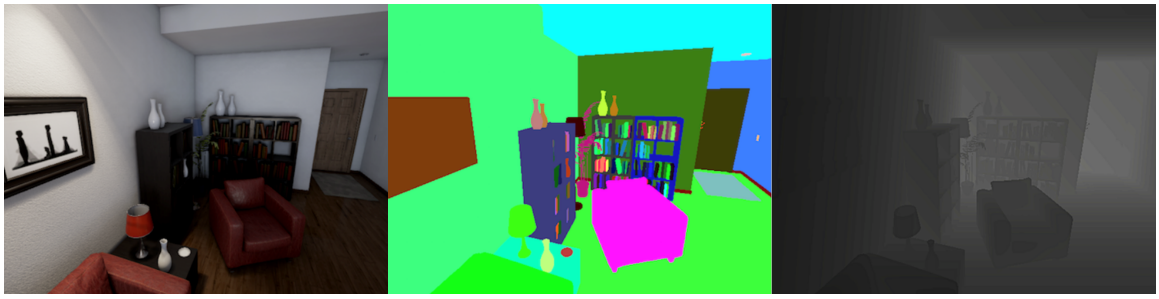


Figure 3-1: Visual modalities from left to right: RGB, Segmentation maps, Depth images

Figure 3-2: Visual modalities from left to right: Bounding boxes, Masking

6. Object masks

All visual modalities except visual flow are fully functional (Figures 3-1 and 3-2), and as are a variety of actions that the agent is able to perform. To further enhance the realism of VirtualCity, we will also incorporate sounds and physics into the environment. Activities and actions emit their respective sounds, for instance, walking, traffic, and weather noise. The in-game physics is built around UE5's Chaos Physics, which allows for water, solid, and collision physics.

VirtualCity is comprised of two main components: A simulated environment running on Unreal Engine [32], and a Python API that allows users to manipulate and interact with the environment as desired. We focused on creating a realistic simulation that can simulate a wide range of objects, scenes, buildings, humans, vehicles, etc. to study various human activities and decision-making tasks in both indoor and outdoor environments. The simulator contains a diverse environment with multiple

configurations of buildings, agents, and everyday objects. The diversity and size of the environment enable both high-level and low-level planning tasks, resulting in the potential to generate tasks that span extended periods of time. We chose to use Unreal Engine [32] as the engine to build VirtualCity due to its photo-realistic rendering capabilities and realistic physics engine. A majority of the work done has been within an apartment setting, as once the indoor environment is configured correctly, we can expand to enabling outdoor interactions. We utilize the high-quality assets from the Unreal Engine marketplace to build this initial environment, which we hope produces less of a sim2real gap when training agents or gathering datasets from VirtualCity.

We focus initially on single-agent tasks, which the VirtualCity simulator and Python API should work together to enable. These include the following:

1. High-level control tasks

2. Language grounding and instruction following

3. 2D vision tasks such as image segmentation and object detection

4. 3D vision and graphics tasks such as 3D reconstruction and 3D generation

3.1.1 Representation

We choose to represent the simulated environment state as a graph, as a graph representation can be easily constructed and interpreted within the environment as well as on the API. The graph is represented in a JSON format, with an initial field indicating the name of the environment that should be loaded, which contains a list of all objects within the environment 1. The “container” field indicates how many containers an object has. A container is essentially a slot to place an item, for instance, a microwave could have one container. As of now, the container field is not in use, as most objects have not had the manual placement of containers. The "id" field indicates the object’s id, this is primarily used when there are multiple objects with the same name to distinguish them. The "name" field indicates the name of

```
{
    "Environment-Name": [
        {
            "conatiner": 0,
            "id": 0,
            "name": "Object1",
            "relation": ["Object1 supported by Object2"],
            "state": ["Open"],
            "transform": [
                "X=0 Y=0 Z=0",
                "P=0 Y=0 R=0",
                "X=1 Y=1 Z=1"
            ]
        },
        {...},...
    ]
}
```

Listing 1: Graph Representation

the object. The "relation" field indicates the object's position relative to the environment, and can have multiple properties, there will be more on this later. The "state" field indicates the object's state and is not required for most objects. For instance, a door can be "open" or "closed", and a microwave could be "on" or "off". Finally, the "transform" field indicates the object's position, rotation, and scale, respectively. This graph structure is easy to compute within the simulator, and can be easily modified by the client by manipulating the nodes.

3.2 API

3.2.1 Interaction

There are two methods in which users can interact with the environment. The first is by playing the simulator as if it were a game, using WASD for movement, space for jumping, and the mouse controlling the camera. This method is rather limited, but it offers users a way to explore the environment. The second option is to interact

with the environment through the API. The API has the following endpoints:

1. `make(environment_id, num_agents)`

   Creates a new default environment of the specified $environment\_id$ with num_agents agents

2. `reset(graph)`

   Resets the current environment with the new graph provided

3. `render(camera_index, image_synthesis, image_width, image_height)`

   Renders an image of the environment at the given camera index with the visual modality type ($image\_synthesis$) and image size specifications

4. `observation(agent_index, radius)`

   Gets the environment graph within a certain radius of the agent

5. `set_action(agent_index, task, coordinate, rotation)`

   Used for high-level control tasks. Commands an agent to perform the given task, with the coordinate and rotation as optional parameters for certain actions such as "grab [object]"

A client can use these commands to interact with a server, which acts as a proxy between the client and the simulator. Each command gets received by the simulator, which handles it as necessary. A majority of interaction with the environment is done through the "set_action" and "reset" commands. Through the "set_action" command, a user is able to control an agent's next action. The current available actions are: Walk, Run, Grab, Place, Put In, Switch on, Switch off, Open, Close, Eat,

Drink, Sit, Stand, Emote ((dance, wave, exercise, point at, look at, etc.). Through a series of these commands, an agent is able to perform most day-to-day activities. For instance, if an agent was tasked with loading the dishwasher, a series of commands could look like: (walk, kitchen), (walk, table), (grab, dish), (walk, dishwasher), (open, dishwasher), (putin, dish, dishwasher). The "reset" command takes in a graph representation of the environment, which the simulator then parses to update the environment. This is used when adding or removing objects from the environment to create a variety of starting states for agents to explore. Another command that is essential to interacting with the simulator is "observation". Through this command, the client is able to get a graph representation of the current environment, enabling the client to make decisions based on what is observed. The "observation" command is what enables our regression planner to make decisions for the agent based on the state of the environment.

3.3 Simulator

3.3.1 Relations and States

When humans perceive a scene, they do not contain a list of spatial coordinates within their minds in order to execute a task. For any given object, we look at its relationship to the objects around it. In that sense, for a task such as "put a plate on a table", we would conclude that the task has been completed when the plate is resting on the table. Similarly, within the context of an agent navigating a scene, it should figure that its task has been completed only when it notices certain relations are satisfied, as opposed to determining the coordinates of an object are where they should be. In order to generate these relations within the simulator, we make use of box-tracing, which is essentially when a box approximately the shape of an object's collision mesh is drawn in a direction and reports any collisions that are made.

When the "observation" command is received by the simulator, it iterates through each object in the current environment to determine the fields for each node in the

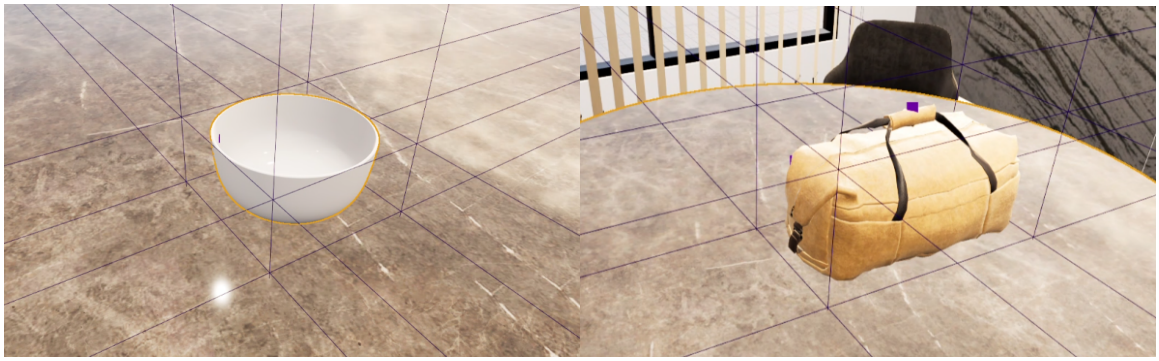


Figure 3-3: Visual representation of the box traces used to form relations.

graph. For each $object_1$, six box-traces are drawn from $object_1$ and the collisions are recorded as seen in Figure 3-3. Four traces are drawn relative to $object_1$'s orientation, in its +X, -X, +Y, -Y directions. Two traces are drawn relative to the world's orientation, one that is upward: +Z, and one that is downward: -Z. All objects except the ground have physics enabled for a more realistic environment, so every object is somehow supported by another. We determine that $object_1$t is supported by the object hit by the -Z trace. If +Z and -Z hit the same $object_2$, then $object_1$ is inside of $object_2$, otherwise, $object_1$ is under the hit object from the +Z trace $object_3$. The rest of the box traces are used to determine a object's proximity to other objects in the scene.

With this relation calculation for each object, agents are able to surmise the state of the environment without actually visualizing object relations from the simulation. Specifically, for the regression planner we implemented, this type of relation is necessary for the agent to determine the state of the environment for planning. Similarly, the state of each object is also necessary for the regression planner. Based on graph state alone, the planner must be able to determine if an appliance is on or off, and if a door is open or closed. To handle this, we added to each object a tag that represents its internal state. Initially, each object is off/closed, but can be modified through the API by commanding the agent to turn on/off or open/close an object through the "set_action" command.

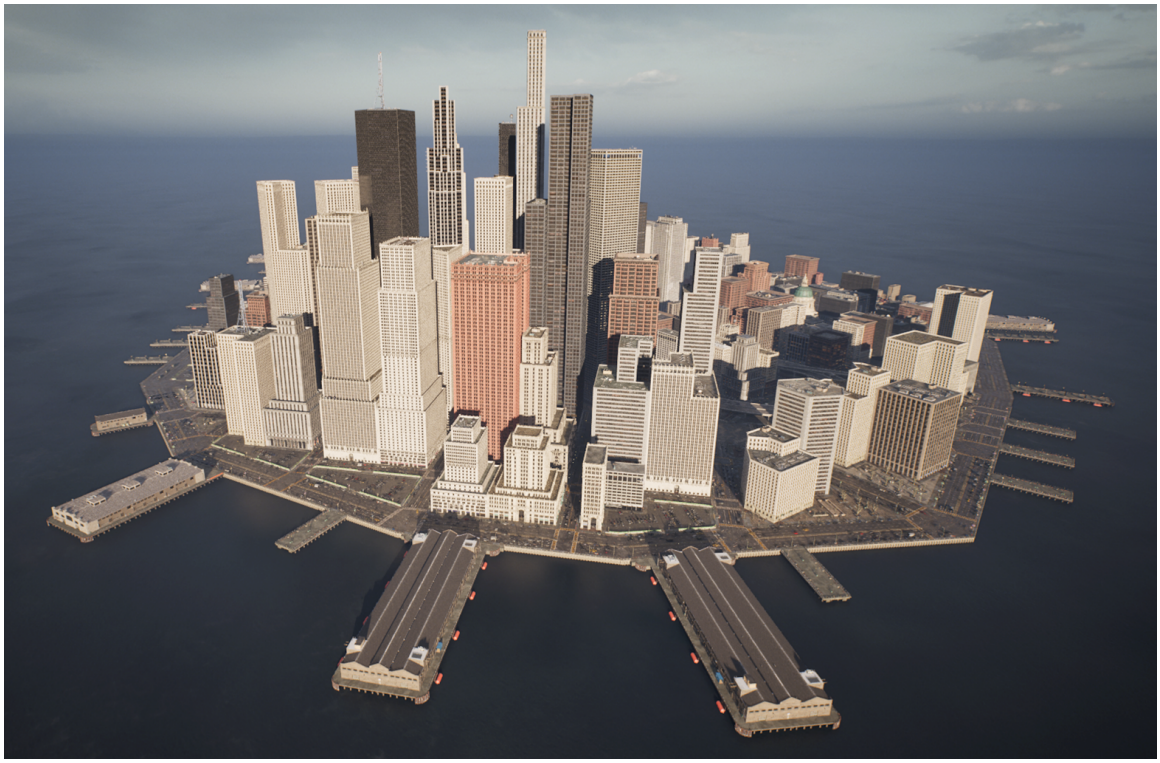


Figure 3-4: Wide view of the whole city environment. Built from the following UE asset packs [23, 27, 26, 20, 24, 19, 25, 22, 21, 8, 9, 6, 7, 5, 16, 28, 1, 18, 12, 15, 17, 11, 14, 2, 10, 4, 13, 3]

3.3.2 Regions

The VirtualCity simulator currently has 2400+ indoor objects, and 4500+ outdoor objects, allowing for a multitude of custom environments that can be setup. Currently, the VirtualCity simulator contains an extensive outdoor city environment (Figures 3-4 and 3-5) with functioning traffic and pedestrians (Figure 3-6). Two indoor environments are now set up, a convenience store (Figure 3-8), and an apartment (Figure 3-7). Through the use of procedural generation, 100000 distinct apartments have been created as of now. This can similarly be used to generate a variety of indoor environments for locations such as offices, restaurants, stores, etc.

Figure 3-5: Close-up of the city

Figure 3-6: City environment from an agent's view

Figure 3-7: Multiple views of an apartment environment

Figure 3-8: Convenience store environment

Simulator	Action	Realistic	Multi-agent	HP	Large-scale
Generative[38]	High/Low-level	No	Yes	Yes	Yes
MineDojo[33]	High/Low-level	No	No	No	Yes
Behavior-1k[42]	High/Low-level	Yes	No	No	No
THOR[35]	High/Low-level	Yes	No	No	No
VirtualHome[39]	High/Low-level	Yes	Yes	Yes	No
VirtualCity	High/Low-level	Yes	Yes	Yes	Yes

Table 3.1: Comparison of VirtualCity with other existing embodied environments. The criteria are as follows: Action space (high and/or low level), realistic environment, multi-agent capabilities, human-like task solver (HP), large-scale environment for high-level tasks.

3.4 Comparison

There are currently many simulated environments used for understanding human activity. Our environment stands out for its ability to perform high-level actions, realistic environments, multi-agent capabilities, support for a human-like agent capable of solving every day human tasks, and large-scale environments for agents to traverse and interact with. A comparison with other existing platforms based on the criteria listed is shown in Table 3.1. With its combination of capabilities, VirtualCity provides a novel and powerful platform for simulating large-scale urban environments and training agents for various tasks, which isn't necessarily possible in existing platforms.

Chapter 4

Model

One of the goals of VirtualCity is to enable easy access to a tool for reinforcement learning and task planning. One method of doing so was outlined in Watch and Help (WAH) [40]. Due to the human-like behavior of agents trained using a hierarchal planner in WAH, we decided to implement a regression planner in the same fashion, where an agent operates on the symbolic representation of its partial observation of the environment. For every step within the environment, an agent has a belief of where objects are located, then generates a list of actions to perform in sequence towards some goal. As the agent observes more of the environment, it is able to update its belief and, consequently, the list of actions. In order for this regression planner to work, we must have some set tasks to generate initial environments, which the planner will then use to generate trajectories. While time did not permit us to train an agent to learn how to complete these tasks, in the future we plan to determine the accuracy of a model trained on this data.

4.1 Tasks

Towards the goal of training an agent to complete a set of tasks, these tasks must be defined. Within the VirtualCity environment, we are constrained by a set of actions that agents are able to perform, as well as the objects within the environment. By providing these constraints to GPT-4 [37], we were able to generate a list of 50

daily household tasks that are able to be performed, as well as 50 outdoor tasks (see appendix B). Each task can be associated with a goal of some sort, where some object of interest has an end state or position relative to the environment. For instance, the task “sit down at the kitchen table and eat breakfast” has three end goals. First, breakfast must be on the kitchen table for the agent to eat, this is represented as “On(breakfast, kitchen table), 1”. Second, the agent must be sitting - “Sit(kitchen chair), 1”. Finally, the agent must eat the breakfast "Eat(breakfast), 1". Every task that was provided can be broken down into these end goals, which are used by the initial environment generator to set up an environment in which these end goals can be satisfied.

4.2 Initial Environments

In order to setup the environment for the human-like agent as described in WAH, we must place objects as necessary within the environment so that every end goal for each task can be completed. To do so, each task is associated with a set of goals, for instance, the task "Place an apple on the table" would be represented by "On(Apple, Table), 1". For each of these end goals, we check to see if an object must be added to the environment. If an object already exists within the environment, there is no need to add it; however, if the object is missing, it must be added to the environment graph. We determine the placement of the object based on where it could potentially be in the real world. For instance, a coat would likely belong on a hangar or in the closet, and a book could be on a coffee table. The initial environment generator chooses a random location from the list of possible locations for each object and adds the object to the graph. Once every object to add has been added to the graph, the generator issues a "reset" command to the environment with this new graph and receives the resulting environment graph observation. Based on the resulting relations of each object, if each object that was added has its correct relation (On(book, coffeetable) etc.), then the environment is ready for the planner. Otherwise, the generator attempts to place the objects again. Once an environment

has been successfully setup, we can then use it for the regression planner.

4.3 Regression Planner

The human-like agent using a hierarchal planner (HP) from WAH attempts to replicate how an actual human would behave when completing a task, and is used as a baseline for training any model that attempts to solve the WAH challenge. Here we delve into more detail about the human-like agent itself, which from this point forward we will refer to as the agent. In VirtualCity, we implement an agent that follows the same procedures as outlined in WAH in order to collect data for a reinforcement learning agent. Figure 4-1 outlines the main components within the agent. For the task of finding where objects could potentially be located, the agent contains a belief module that informs the planner where an object could potentially be. Every object within the environment contains a list of possible locations within the belief module, each with an equal chance of being the object's location, enabling the agent to attempt to find objects within the environment much as an actual person without any knowledge of the environment would.

Initially, the agent receives the goal state as well as an observation of the environment. Based on the agent's current location within the environment, it will receive a partial observation that contains only objects found in the room it's in. By combining the belief and partial observation, where the partial observation is the true state of the environment, the agent generates a state in which it samples an object's potential location and adds it to the state graph. To keep the belief states consistent between iterations of this process, we only resample object positions that have been proven to be false (e.g., if a plate was believed to be on the kitchen counter, but was actually not there, the plate position would be resampled). This then brings us to the planner module, which determines which actions the agent should take in order to accomplish the goal.

Given the belief state of the agent, the planner will then generate a plan for attaining a given goal state. First, based on the belief state of the agent, we generate

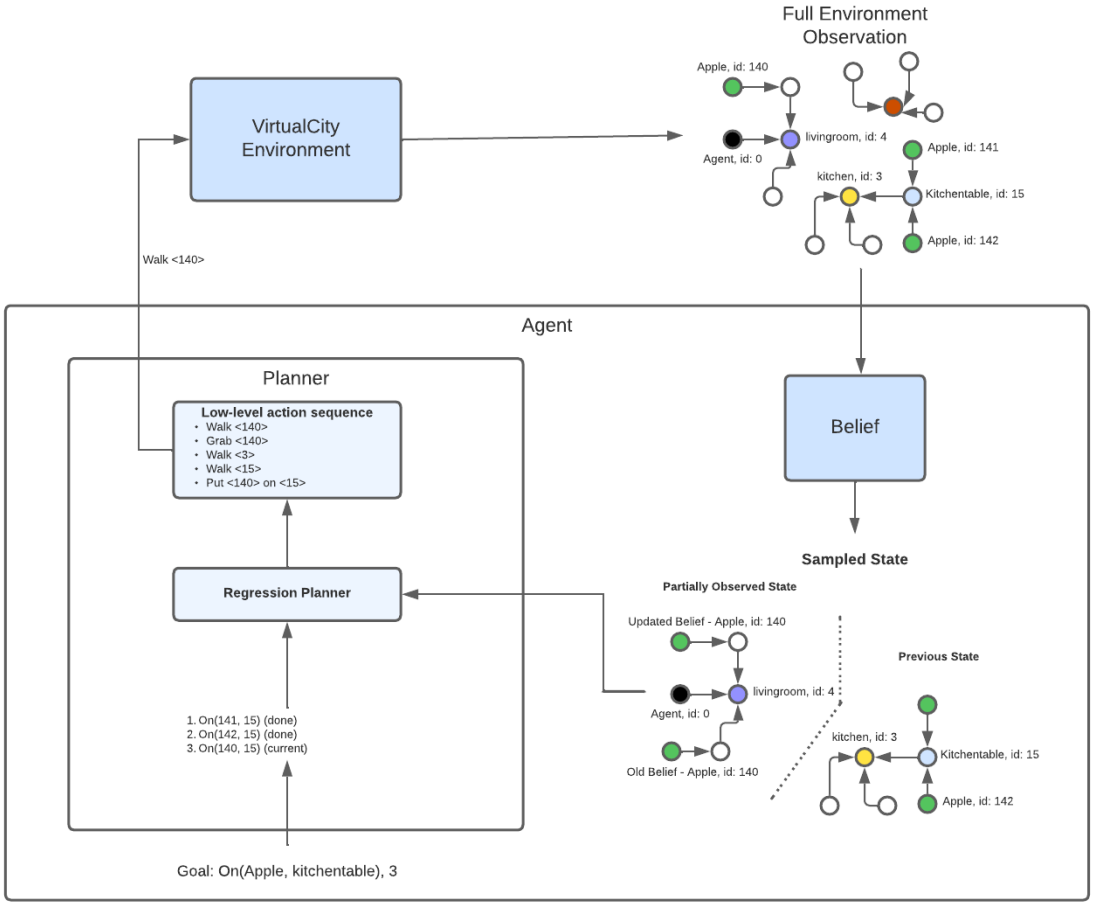


Figure 4-1: This Figure shows the high-level diagram of the human-like agent. Starting from the Full Environment Observation, we can see that the agent is located within the livingroom. Since the agent is within the livingroom, its partial observation of the environment only includes nodes within the room it is in. The agent's previous belief stated that the apple with id 140 is located on the coffeetable; however, the observation shows the apple on another table. The Sampled State combines the agent's previous observations with its current as well as its belief of where objects are, then sends this information to the planner to choose the agent's next steps.

a series of subgoal predicates that would satisfy the overall goal. For instance, if the agent's goal is to place three apples on the kitchen table, represented by "On(apple, kitchentable), 3", we would generate the specific subgoals "On(140, 15), On(141, 15), On(142, 15)", where the kitchentable id is 15, and apple the ids are 140, 141, and 142. By adding extra predicates for certain tasks, we are able to define a list of subgoals to achieve in order to accomplish the overall goal in a realistic fashion. For instance, if the goal of a task is to place an object that is in the fridge somewhere else, we add a predicate "Open(fridge)". Finally, the regression planner generates a list of actions based on each subgoal. It does so by working backwards from the end state of the subgoal, and checking whether certain conditions are met. For instance, subgoal "On(141, 15)" could lead to a sequence of actions such as ["walk <kitchen.id>", "walk <kitchencounter.id>", "grab <apple.141>", "walk <kitchentable.15>", "place <apple.141> <kitchentable.15>"]. The agent then performs the first action in the generated sequence, and repeats these steps. Once the agent receives an updated observation after an action, the regression planner takes into account the current state of the graph and updates the sequence of actions accordingly.

At each iteration, the agent checks whether or not a subgoal has been completed based on relations within the observation graph. Once it observes a completed subgoal, the agent then begins the whole process once again with a new subgoal, if one is available. Once the agent has finished every subgoal, it then records the list of actions it took to complete the task. As the graph representation of the agent's observations are recorded at each action step, we are able to perform some post-processing on the agent's actions to ensure that the agent has completed the task successfully. This processing checks if there were duplicate actions, if the object states and agent observations were correct, and if the graph relations are as expected. Through this process, we are able to verify the agent's ability to complete a task.

Chapter 5

Experiments

5.1 Performance

As the process of generating an agent's plan using the HP was the same as in VirtualHome [39]), the planner's performance is comparable to VirtualHome's hierarchical planner (HP). On average, the HP from VirtualHome was able to generate the agent's next action in 0.05 seconds. We tested the HP in VirtualCity only within the apartment environment, as the size of the graph is similar to that of apartments in VirtualHome. On average, the HP in VirtualCity was able to decide on a new action in 0.05s. With further improvements to graph handling, the action generation time could be reduced. However, it still seems that the major bottleneck for completing a task lies within the environment itself. We have also compared the environment interaction functions within both VirtualHome and VirtualCity in Table 5.1. As mentioned in Chapter 1, we believed that the performance benefits of using Unreal Engine should allow us to work with larger areas for agents to navigate than in VirtualHome. The commands were tested in the apartment setting, showing a massive improvement in initial build time with the executable version of VirtualCity, as well as significant improvements when using the *make* and *reset* commands. However, within an environment consisting of the entire city, we expect significant increases in the command run times.

Command	VirtualHome	VirtualCity
Starting the Executable	4.56-5.07 seconds	0.42-0.95 seconds
Make('Apartment_0')	1.88-3.09 seconds	0.22-0.27 seconds
Reset()	1.30-2.15 seconds	0.17-0.27 seconds

Table 5.1: Execution time for same commands in both VirtualHome and VirtualCity (Tested on a computer with AMD 5900x processor, RTX 3090 GPU, on Windows 11.

5.2 Tasks

To test the accuracy of the human-like agent as well as several baselines, we generated a set of 145 tasks to complete. These tasks were based on a combination of tasks from VH and GPT-4 [37]. The set of 145 tasks was generated from a base of 20 tasks (full list in appendix B). Since each of the base tasks can have interchangeable objects of interest, we replaced the objects from some of the base tasks with others that could also fit the description of the task. For instance, the task "put_fridge" originally uses the objects (SM_Glass_Milk, SM_Kitchen_Bread_Ingir, SM_Breakfast, SM_Jug3), but can be replaced by (SM_Apple, SM_Bread, SM_Wine) to have a similar task. More one-dimensional tasks such as "Close the Blinds to Block the Sun" were only used a few times since there is less room for object replacement.

5.3 Baselines

We set up two different baseline models besides the human-like agent for testing task completion. The first was a random model in which, for each task, up to 100 actions were generated towards completing the goals. To generate an action, the random model samples uniformly from the list of available actions in VirtualCity, then samples a random object from all objects of interest within the environment. The objects of interest were generated for each task based on the initial graph and goals for the task. Since the actions generated by the random model are often infeasible, if the action cannot be performed, then we ignore it in the simulator. The random model acts as a control for our evaluation.

The second baseline used was GPT-4. For each of the 145 tasks and corresponding

initial graphs, we give GPT-4 the subgoals for the task along with the initial graph of the environment. Since there are preconditions within VirtualCity for some actions to work (grabbing and placing objects require the agent to be in front of the object being grabbed or the destination placing the object), we also define the appropriate steps necessary for the agent to be able to grab/place objects correctly. We then ask GPT-4 to generate a sequence of actions that it thinks would lead to the completion of the subgoals. As GPT-4 is quite advanced, we expect it to be able to understand the subgoals for each task and how they should be completed.

Finally, the planner is already integrated into VirtualCity, so we simply provide it with the 145 tasks and evaluate whether or not they have been completed.

5.4 Evaluation

For each of the baselines as well as the human-like agent, we generated initial environments as described in Chapter 4. Then, we ran the VirtualCity simulator on the actions generated for each of the methods. The results for each method is described in Table 5.2. It should be noted that when a task is not completed, the steps to completion are set at 100 (the maximum number of steps for each task). As expected, the random model had a low completion accuracy for the tasks. Of the tasks that it was able to complete, most of them were simple tasks that had few objects of interest, such as "Close the Blinds to Block the Sun", increasing the odds that a sequence of valid actions could occur. The actions generated by GPT-4 had a high probability of accomplishing each task. Four of the failure cases for GPT-4 were due to its misinterpretation of the subgoals, resulting in actions that attempted to turn static objects on/off, when, in reality, the objects should be placed on another. Two of the failure cases for GPT-4 were due to it only producing actions to accomplish a subset of the subgoals. While the planner had a higher completion accuracy than GPT-4, the average number of steps it takes to complete a task is slightly higher than GPT-4. While this means that the list of actions generated by GPT-4 is more efficient than the planner's, the actions generated by GPT-4 may not represent how

	Random	GPT-4	Planner
Goal Completion Accuracy	16.55%	77.93%	82.07%
Average Steps	95.69 steps	21.90 steps	27.61 steps
Average Steps for Successful Tasks	73.96 steps	10.83 steps	11.80 steps

Table 5.2: Results from running actions generated by the random baseline, GPT-4, and the Planner. Goal completion accuracy describes how often the method managed to accomplish the task; average steps indicate how many steps the method took for the tasks on average; and average steps for successful tasks indicate how many steps were required on average for only the tasks that were successfully completed.

a human would actually approach this task. Since GPT-4 is supplied with the initial environment graph, it already knows the locations of every object of interest, and is able to generate a sequence of actions to most efficiently finish the task without exploring the environment. If a human were placed in an unknown environment, they may have a guess as to where objects are, but they must first explore the potential locations of the object before finding it. The planner emulates human behavior with its belief module, resulting in more realistic behavior.

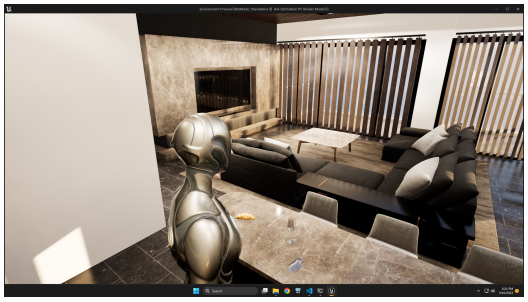
On inspection of the failure cases, it seems that there are some issues with object placement, causing objects being placed to spawn inside of another object, which causes it to eject to locations out of bounds. This would then cause the agent to attempt to find the missing object, which isn't reachable. Another issue that was found originated from objects being spawned in unreachable areas due to a bug in the initial environment generator. For the 26 tasks that the planner failed, every one of them stemmed from an issue within the environment itself. Similarly, out of the 32 tasks that GPT-4 failed, 26 of them failed due to an environment error. As we fix these bugs, we expect that the planner will achieve close to 100% accuracy on task completion.

We initially focused on ensuring indoor environments were fully functional before working on outdoor tasks. As we are just nearing the completion of the indoor environments, we have not been able to run any experiments on outdoor or large-scale tasks. However, once the issues within the indoor tasks are solved, transitioning to the planner for outdoor tasks should be simple, as the functionality of the planner and initial environment generator are similar for indoor and outdoor regions.

5.5 Qualitative Analysis

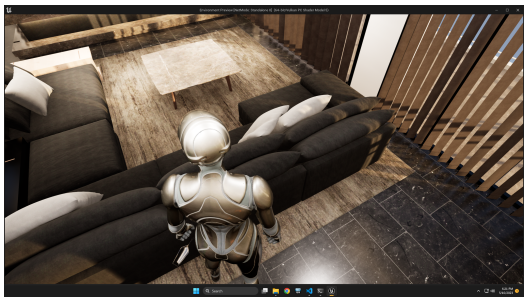
In Figures 5-1 and 5-2 we show a few sequences of actions generated by the human-like agent for a given goal. From the sequence of actions, we can mostly determine whether or not a goal has been completed or not. In Figure 5-1, you can see that the wine glass has been placed on the table in the final sequence, indicating that the agent's objective has been completed. In Figure 5-2, it is harder to tell if the goal has been accomplished since we have not gotten animations configured for some objects, such as the stove. There are some limitations to the current simulation. Notice that the wine glass that the agent is holding in Figure 5-1 clips through the agent in an unrealistic fashion. Another limitation currently is that if the agent wants to walk towards an object in a room it has already visited, it first walks to the room, then to the object. This often leads to backtracking, which would not happen with a real human performing the task.

Walk kitchen



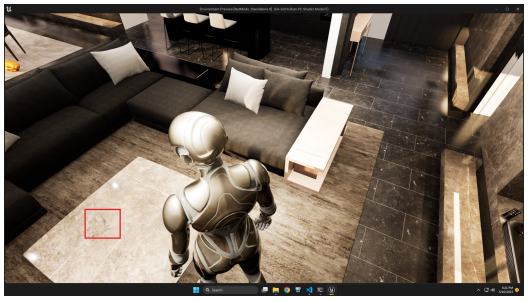
Walk SM_Glass

Grab SM_Glass



Walk livingroom

Walk SM_L_Table

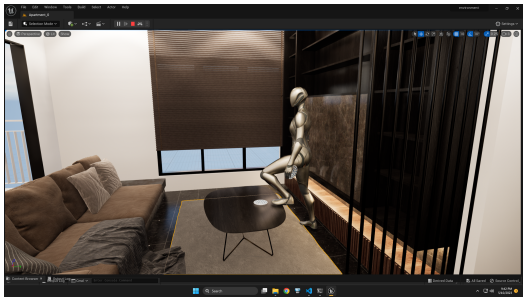


Place SM_Glass on SM_L_Table

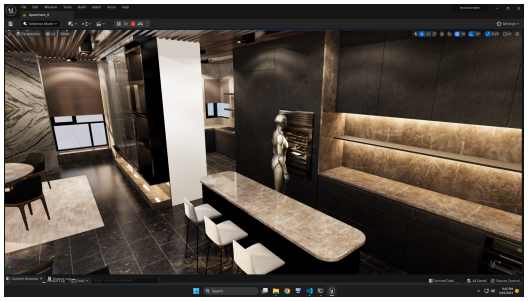
Figure 5-1: Sequence of actions for the goal "On(SM_Glass, SM_L_Table)"

Walk livingroom_2

Walk SM_Plate

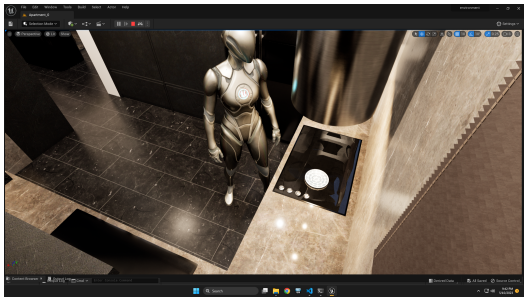


Grab SM_Plate

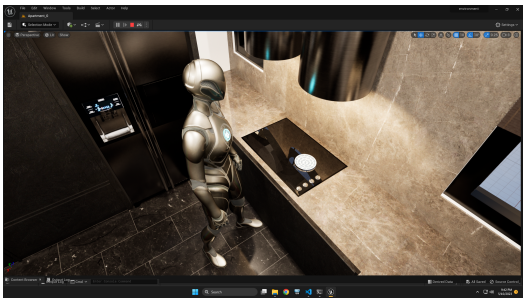


Walk kitchen

Walk SM_Cooker



Place SM_Plate on SM_Cooker



Switchon SM_Cooker

Figure 5-2: Sequence of actions for the goal "On(SM_Plate, SM_Cooker)"

Chapter 6

Conclusion

6.1 Contributions

This thesis describes VirtualCity, which was created in an attempt to address the issues of existing multi-agent simulators. VirtualCity is both realistic and performant, enabling efficient learning of human behavior. The contributions of VirtualCity extend beyond the existing embodied environments, which mostly focus on small-scale or low-level control tasks. By enabling a larger environment in which agents can navigate, more complex and long-term tasks can be learned. With a realistic environment and multi-modal observations of the environment, agents can potentially learn information about their environments from images in the agent's views.

Through the creation of a human-like agent using a regression planner, we have taken the first step towards training a reinforcement learning agent to complete day-to-day human tasks. We have demonstrated that the planner is able to complete most tasks it is assigned to complete, which serves as a demonstration of the capabilities of the VirtualCity Simulator. In summary, VirtualCity serves as a significant step towards creating a real-world agent capable of human activities. The developments in this project can lead to further research and advancements in simulating real-world scenarios, making it possible to improve autonomous agents' performance in the real world.

6.2 Future Work

The work in this thesis has shown the current capabilities of VirtualCity; however, more work on features, animations, and training setup has yet to be completed. Immediate future work includes fixing the environment bugs that cause sequences of actions that should result in completing a task to fail. Once that is completed, we will generate task definitions for a larger set of tasks so that the human-like agent is able to generate enough action trajectories to train a reinforcement learning agent. This should serve as a baseline for our simulator's performance as compared to VirtualHome. Next, as many of the animations for agent actions have yet to be incorporated, we will ensure that every action the agent performs is done so with a corresponding animation.

In VirtualCity's current state, we have an indoor environment that is near fully functioning, however, we don't have full control over the outdoor environment as of now. The next major step would be to expand the initial environment generation and regression planner code to accommodate outdoor environments. Our initial goal with VirtualCity was to create an environment that enables RL of large-scale human activities, and to do so, we must have our planner work in the outdoor environment. Furthermore, there must be seamless transitions between indoor and outdoor environments. This would ensure that any agent that is performing a large-scale task can do so without having issues traversing between the indoor and outdoor environments.

Appendix A

Asset Packs Used

Below is a list of URLs from the assets used within the VirtualCity environment.

1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-decoration-pack-1
2. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-storages-pack-1
3. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-sports-equipment-pack-2
4. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-backyard-pack-2
5. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-entertainment-pack-1
6. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-backyard-pack-1
7. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-high-tech-pack-1
8. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-chairs-tables-pack-1

9. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-boats-pack-1
10. https://www.unrealengine.com/marketplace/en-US/product/city-sample-vehicles
11. https://www.unrealengine.com/marketplace/en-US/product/city-sample-buildings
12. https://www.unrealengine.com/marketplace/en-US/product/city-sample-crowds
13. https://www.unrealengine.com/marketplace/en-US/product/city-sample
14. https://www.unrealengine.com/marketplace/en-US/product/metahumans
15. https://www.unrealengine.com/marketplace/en-US/product/v-residences
16. https://www.unrealengine.com/marketplace/en-US/product/alila-suite
17. https://www.unrealengine.com/marketplace/en-US/product/singapore-condominium
18. https://www.unrealengine.com/marketplace/en-US/product/contemporary-suites
19. https://www.unrealengine.com/marketplace/en-US/product/lyra
20. https://www.unrealengine.com/marketplace/en-US/product/city-subway-train-modular
21. https://www.unrealengine.com/marketplace/en-US/product/3ac6699c15a34926afc47bfbe15f4fc5
22. https://www.unrealengine.com/marketplace/en-US/product/31797bf6fa0545d590e3bb17d0968de
23. https://www.unrealengine.com/marketplace/en-US/product/8162a702d7c747e9ac544dff38af78c8
24. https://www.unrealengine.com/marketplace/en-US/product/cca-subway-train-terminal
25. https://www.unrealengine.com/marketplace/en-US/product/convenience-store
26. https://www.unrealengine.com/marketplace/en-US/product/archvis-interior-rendering
27. https://quixel.com/bridge

Appendix B

Tasks

List of indoor tasks generated by ChatGPT

1. ChatGPT tasks given objects in environment and action constraints:
2. Walk to the kitchen and grab a glass from the cabinet.
3. Sit on the sofa and watch TV.
4. Stand up and walk to the door to greet a guest.
5. Open the fridge and grab some leftovers.
6. Close the window blinds to block the sun.
7. Turn on the ceiling light and read a book.
8. Sit at the desk and use the computer.
9. Walk to the bedroom and switch off the lamp.
10. Grab a towel and take a shower.
11. Walk to the closet and grab a coat.
12. Put the dirty dishes inside the dishwasher and turn it on.
13. Grab a broom and sweep the floor.

14. Walk to the living room and turn off the TV.
15. Sit at the dining table and have a meal.
16. Walk to the garage and grab a tool from the shelf.
17. Open the window and let some fresh air in.
18. Close the door to keep the noise out.
19. Stand up and stretch after sitting for a long time.
20. Walk to the balcony and water the plants.
21. Grab a book from the shelf and read it in bed.
22. Sit on the stool and tie your shoes.
23. Turn off the ceiling fan to conserve energy.
24. Walk to the laundry room and start a load of laundry.
25. Place the groceries on the kitchen counter and unpack them.
26. Walk to the front door and wave goodbye to a friend.
27. Walk to the refrigerator and open it.
28. Run to the bedroom and sit on the bed.
29. Grab a glass and fill it with water from the kitchen sink.
30. Stand up and switch off the lights in the living room.
31. Walk to the front door and open it.
32. Sit at the dining table and grab a fork.
33. Stand up and close the window in the bathroom.
34. Open the laptop and start typing.

35. Switch off the TV in the entertainment center.
36. Walk to the laundry room and grab a basket.
37. Put the dirty clothes in the washing machine and start it.
38. Sit at the desk and grab a pen and paper.
39. Stand up and switch on the lamp in the bedroom.
40. Walk to the bookshelf and grab a book.
41. Open the book and start reading.
42. Close the door to the balcony.
43. Sit on the couch and grab a pillow.
44. Stand up and switch off the fan in the living room.
45. Walk to the kitchen and grab a knife.
46. Put the vegetables in the pot and place it on the stove.
47. Walk to the mirror in the bathroom and emote by smiling.
48. Open the cabinet in the hallway and grab a jacket.
49. Sit in the car and switch on the engine.
50. Drive to the grocery store and grab a shopping cart.
51. Walk around the store and grab some food items.

Outdoor tasks generated by GPT-4

1. Walk to the park and have a picnic with friends.
2. Run errands and stop by the grocery store, pharmacy, and bank.
3. Grab a book from the shelf and read it in bed.

4. Place the laundry in the washing machine and start it.
5. Sit at the dining table and enjoy a home-cooked meal.
6. Stand up and stretch after sitting for a long time.
7. Open the windows and let in some fresh air.
8. Close the blinds to block the sun.
9. Put the dirty dishes in the dishwasher and turn it on.
10. Switch off the lights in the living room to conserve energy.
11. Walk to the fridge and grab a cold drink.
12. Emote by waving at a neighbor across the street.
13. Walk to the closet and grab a coat before heading out.
14. Open the car door and get inside.
15. Drive to the beach and go for a swim.
16. Walk to the movie theater and catch a new film.
17. Place the groceries in the fridge and pantry.
18. Sit at the desk and use the computer.
19. Stand up and switch on the lamp in the bedroom.
20. Walk to the bathroom and take a relaxing bath.
21. Grab a towel and dry off after a shower.
22. Place the tools back in the toolbox after finishing a project.
23. Open the curtains and let the sunlight in.
24. Close the garage door after parking the car.

25. Sit on the couch and watch TV.
26. Stand up and turn off the fan in the living room.
27. Walk to the park and play with your dog.
28. Grab a basketball and shoot some hoops at the court.
29. Place the dirty clothes in the laundry hamper.
30. Switch off the ceiling fan to conserve energy.
31. Walk to the front door and open it for a delivery.
32. Grab a broom and sweep the kitchen floor.
33. Sit on the patio and enjoy a cup of coffee.
34. Stand up and turn on the air conditioning to cool down.
35. Walk to the bedroom and make the bed.
36. Grab a book from the library and start reading.
37. Place the dirty dishes in the sink and wash them by hand.
38. Open the oven and check on the food cooking inside.
39. Sit at the table and play board games with family.
40. Stand up and turn on the heater to warm up.
41. Walk to the closet and grab a hat before heading outside.
42. Grab a shovel and do some gardening.
43. Place the toys back in the toy chest after playing.
44. Open the windows and enjoy the sound of rain.
45. Close the door to keep out the cold air.

46. Sit at the piano and play a song.

47. Stand up and open the blinds to let in the sunlight.

48. Walk to the corner store and grab some snacks.

49. Grab a coat and go for a walk in the park.

50. Emote by dancing to your favorite song.

Appendix C

Interactable Objects

- 50 x Chairs
- 8 x Stools
- 86 x Tables
- 37 x Dining Table
- 15 x High Table
- 33 x Low Table
- 2 x Printer
- 2 x Audio Amplifier and Controller
- 9 x Camera
- 10 x Computer/Laptop/Tablet
- 1 x Drone
- 1 x DVD Player
- 2 x Graphic Tablet
- 3 x Headset/Earphone

- 2 x Micro
- 3 x PC Keyboard
- 4 x PC Mouse
- 7 x Speaker
- 10 x Screen/TV
- 1 x Softbox
- 2 x Tripod
- 1 x Turntable
- 1 x Virtual Headset
- 2 x Webcam
- 44 x Libraries
- 58 x Bathroom storage
- 47 x Professional kitchen storages and equipments
- 17 x Sport Machines
- 31 x Sport Equipments
- 41 x Sport Accessories
- 2 x Air hockey tables
- 3 x Arcade games
- 2 x Foosball table
- 1 x Console
- 2 x Controllers

- 1 x Dart board
- 1 x Hoverboard
- 4 x Joysticks
- 1 x Jukebox
- 2 x Ping pong tables
- 4 x Pool tables
- 4 x Strollers
- 49 x Kids' games for backyard
- 39 x Toy games for interior
- 24 x Decorations
- 52 x Books
- 10 x Candles
- 6 x Frames
- 40 x KitchenWare
- 10 x Mirrors
- 11 x Vases
- 4 x Backyard Campfire
- 3 x Backyard Firewood
- 1 x Backyard Pizza Oven
- 3 x Barbecue
- 4 x Buoy

- 2 x Carport
- 1 x Composter
- 1 x Diving Board
- 5 x Garden Accessories
- 3 x Garden Hose
- 3 x Garden Shed
- 2 x Garden Storage
- 1 x Gas Tank
- 3 x Jacuzzi
- 1 x Kennel
- 2 x Ladder
- 3 x Lawnmower
- 6 x Pool
- 1 x Pool Goal
- 1 x Pool Volleyball
- 1 x Solar Shower
- 2 x Watering Can
- 1 x Wheelbarrow
- 24 x Backyard chairs
- 20 x Backyard sofas
- 27 x Backyard tables

- 13 x unique driveable vehicles
- 97 x food items

Bibliography

[1] alila-suite. https://www.unrealengine.com/marketplace/en-US/product/alila-suite.

[2] Apartment tech props. https://www.unrealengine.com/marketplace/en-US/product/31797bf6fa0545d590e3bb17d0968dea.

[3] archvis-interior-rendering. https://www.unrealengine.com/marketplace/en-US/product/archvis-interior-rendering.

[4] cca-subway-train-terminal. https://www.unrealengine.com/marketplace/en-US/product/cca-subway-train-terminal.

[5] city-sample. https://www.unrealengine.com/marketplace/en-US/product/city-sample.

[6] city-sample-buildings. https://www.unrealengine.com/marketplace/en-US/product/city-sample-buildings.

[7] city-sample-crowds. https://www.unrealengine.com/marketplace/en-US/product/city-sample-crowds.

[8] city-sample-vehicles. https://www.unrealengine.com/marketplace/en-US/product/city-sample-vehicles.

[9] city-sample-vehicles. https://www.unrealengine.com/marketplace/en-US/product/city-sample-vehicles.

[10] City street props. https://www.unrealengine.com/marketplace/en-US/product/8162a702d7c747e9ac544dff38af78c8.

[11] city-subway-train-modular. https://www.unrealengine.com/marketplace/en-US/product/city-subway-train-modular.

[12] contemporary-suites. https://www.unrealengine.com/marketplace/en-US/product/contemporary-suites.

[13] convenience-store. https://www.unrealengine.com/marketplace/en-US/product/convenience-store.

[14] Freeway props. https://www.unrealengine.com/marketplace/en-US/product/3ac6699c15a34926afc47bfbe15f4fc5.

[15] lyra. https://www.unrealengine.com/marketplace/en-US/product/lyra.

[16] metahumans. https://www.unrealengine.com/marketplace/en-US/product/metahumans.

[17] Quexel bridge. https://quixel.com/bridge.

[18] singapore-condominium. https://www.unrealengine.com/marketplace/en-US/product/singapore-condominium.

[19] twinmotion-backyard-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-backyard-pack-1.

[20] twinmotion-backyard-pack-2. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-backyard-pack-2.

[21] twinmotion-boats-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-boats-pack-1.

[22] twinmotion-chairs-tables-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-chairs-tables-pack-1.

[23] twinmotion-decoration-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-decoration-pack-1.

[24] twinmotion-entertainment-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-entertainment-pack-1.

[25] twinmotion-high-tech-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-high-tech-pack-1.

[26] twinmotion-sports-equipment-pack-2. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-sports-equipment-pack-2.

[27] twinmotion-storages-pack-1. https://www.unrealengine.com/marketplace/en-US/product/twinmotion-storages-pack-1.

[28] v-residences. https://www.unrealengine.com/marketplace/en-US/product/v-residences.

[29] Nvidia physx 4.5 and 5.0 sdk, Apr 2023.

[30] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects, 2022.

[31] Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Jordi Salvador, Kiana Ehsani, Winson Han, Eric Kolve, Ali Farhadi, Aniruddha Kembhavi, and Roozbeh Mottaghi. Procthor: Large-scale embodied ai using procedural generation, 2022.

[32] Epic Games. Unreal engine.

[33] Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge, 2022.

[34] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[35] Eric Kolve, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: an interactive 3d environment for visual AI. *CoRR*, abs/1712.05474, 2017.

[36] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Matt Deitke, Kiana Ehsani, Daniel Gordon, Yuke Zhu, Aniruddha Kembhavi, Abhinav Gupta, and Ali Farhadi. Ai2-thor: An interactive 3d environment for visual ai, 2017.

[37] OpenAI. Gpt-4 technical report, 2023.

[38] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior, 2023.

[39] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs, 2018.

[40] Xavier Puig, Tianmin Shu, Shuang Li, Zilin Wang, Yuan-Hong Liao, Joshua B. Tenenbaum, Sanja Fidler, and Antonio Torralba. Watch-and-help: A challenge for social perception and human-ai collaboration, 2021.

[41] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks, 2019.

[42] Sanjana Srivastava, Chengshu Li, Michael Lingelbach, Roberto Martín-Martín, Fei Xia, Kent Vainio, Zheng Lian, Cem Gokmen, Shyamal Buch, C. Karen Liu, Silvio Savarese, Hyowon Gweon, Jiajun Wu, and Li Fei-Fei. Behavior: Benchmark for everyday household activities in virtual, interactive, and ecological environments, 2021.