

POLITECNICO DI MILANO
Scuola di Ingegneria Industriale e dell'Informazione
Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria



Asymmetrical multiplayer videogame

Relatore: Prof. Marco Gribaudo

Tesi di Laurea di:
Jacopo Grandi, Matr. 947077

Anno Accademico 2020/2021

Abstract

Games are a powerful tool to improve the player's mental and physical skills. The skills we are interested in training are speed and efficiency of communication, planning and problem solving. They are useful in situations such as a bomb defusal, cave rescues and thick fog navigation, where a team needs remote support to make good decision. We designed and implemented a complete videogame with asymmetrical multiplayer in order to train such skills.

Contents

1	Introduction	3
1.1	Applicable scenarios	3
2	State of the art	4
2.1	Asymmetrical Multiplayer	4
2.2	Similar Games	4
3	Game Design	8
3.1	Traffic simulation	8
3.2	Items	9
3.3	Master's Information	9
3.4	Interface design and menus	10
4	Implementation	13
4.1	Level design and 3D models	13
4.2	UDP Sockets with fragmentation and optional retransmission . .	16
4.3	Game Architecture	17
4.4	Synchronization	18
4.4.1	Dead reckoning and latency hiding	18
4.4.2	Advantages over HLAPI [6]	19
4.5	Agent Based Traffic Simulation	20
4.5.1	Definition and Assumptions	20
4.5.2	Implementation	23
4.5.3	Optimization	24
4.5.4	Traffic Synchronization	25
4.6	Kinematic Bycicle Model [10]	26
4.6.1	Player movement	27
4.7	Radio	28
4.8	Video	30
4.9	Editor Tools	30
5	Testing	32
5.1	Unit Testing	32
5.1.1	Test Driven Development [8]	32
5.2	Integration Testing	32
5.3	Clumsy [3]	32
6	Conclusion	33

1 Introduction

Efficient communication and skill development The game is focused in training the communication, planning and problem solving skills of the players. The games puts the players under time pressure and requires a significant amount of information to be exchanged quickly. It also forces a half duplex communication protocol emulating a radio, such that only one player may speak at a time. The courier players will also have to solve space navigation puzzles, obstacle avoidance and local route planning; others will have to make a general plan, assign tasks and deal with multitasking.

Keep Talking and Nobody Explodes - Elevator of Doom [5] A group of players of Keep Talking and Nobody Explodes completed a challenge called the "Elevator of Doom", which is a exceptionally difficult level. To finish in the 20 minutes time limit, the players spoke quickly in a condensed and clear manner to exchange the puzzles state and the solution. While the players learned how to beat the game, they also developed their communication skills. The level reached by these players is an example of the possibilities offered by the medium.

1.1 Applicable scenarios

This skills are useful in environments where the team has no way to take decisions based on their local space and has to rely on another team that has the global picture to make decision for them. Examples of scenarios in such environments are cave exploration or cave rescues, firefighting operations in thick fog, bomb defusal and warfare operations.

2 State of the art

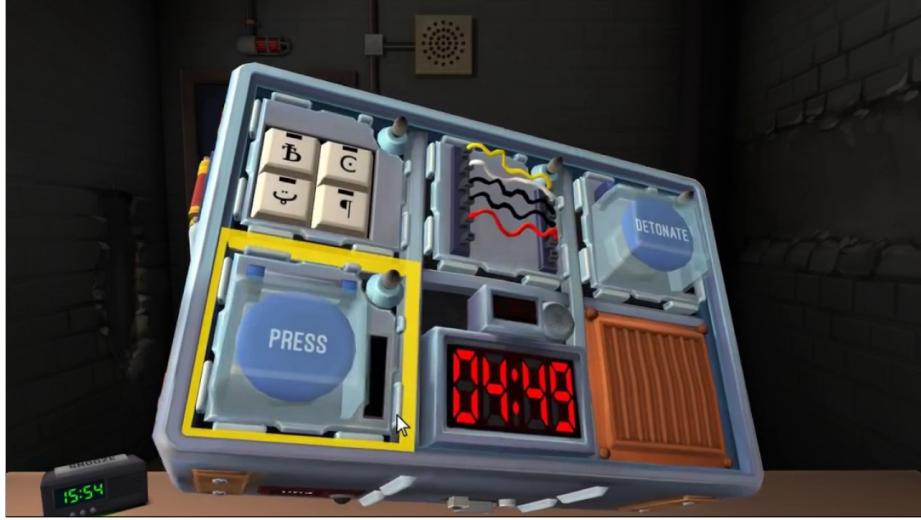
Asymmetrical games are not rare. The vast majority of games have some element of asymmetry, being the character loadout in the case of Battlefield, the factions in the case of Starcraft or the map in the case of Age of Empires. However, the more a game is asymmetric the harder is to balance and produce. Games with a lot of asymmetry are rare. They need to be implemented multiple times, because every player can act differently.

2.1 Asymmetrical Multiplayer

A game has asymmetrical multiplayer when the players play the game differently. There are multiple levels of asymmetry: from a slight imbalance in the mechanics to a completely separate set of rules. Keep Talking and Nobody Explodes is the most asymmetrical game in the cited games: a player just has a manual. The other two games have a lesser degree of imbalance: in Unrailed the roles and tasks can be exchanged during the games and in amongus every player is controlling a character and interacting in the game's map.

2.2 Similar Games

Keep Talking and Nobody Explodes [4] The game consists in defusing a ticking bomb by solving puzzles attached to it. It's a two player game, one player is the defuser and the other has a bomb-defusal manual. The players have to exchange the puzzle state and solutions.



Keep Talking and Nobody Explodes v. 1

On the Subject of Who's on First

This contraption is like connecting out of a scratchy cootchy routine, which might be funny if it's ever connected to a bomb. I'll keep this brief as words only complete sentences.

1. Read the display and use step 1 to determine which button label to read.
2. Using this button label, use step 2 determine which button to push.
3. Repeat until the module has been disarmed.

Step 1:
Based on the display, read the label of a particular button and proceed to step 2:

YES	FIRST	DISPLAY	CHAT	SAYS	NOTHING
BLANK	NO	LED	LEAD	READ	
RED	REED	LEED	HOLD ON	YOU	YOU ARE
YOUR	YOU'RE	UR	THEIR	THEIRER	THEIRKA
THEY ARE	SHE	O	ONE		

DISPLAY

TEST TEST
TEST TEST
TEST TEST

Who's on First

Keep Talking and Nobody Explodes v. 1

Who's on First

Step 2:
Using the label from step 1, push the first button that appears in its corresponding list:

READY	TEA, CHAT, WHAT, MIDDLE, LEFT, PRESS, RIGHT, BLATE, READY, SO, FIRST, URH, SOTHEHO, WAIT
FIRST	LEFT, CHAT, TEA, MIDDLE, NO, RIGHT, HOTTING, URH, WAIT, BLATE, BLATE, WHAT, PRESS, FIRST
NO	BLATE, URH, WAIT, FIRST, WHAT, READ, SIGHT, TBA, POTTERO, LEFT, PRESS, CHAT, NO, MIDDLE
BLATE	WAIT, SIGHT, CHAT, MIDDLE, BLATE, PRESS, READ, HOTTING, NO, WAIT, LEFT, URH, TBA, FIRST
NOTHING	URH, SIGHT, CHAT, MIDDLE, TBA, BLATE, NO, PRESS, LEFT, WAIT, FIRST, POTTERO, READY
TEST	CHAT, SIGHT, URH, MIDDLE, FIRST, WAIT, PRESS, READ, POTTERO, TBA, LEFT, BLATE, NO, WAIT
WHAT	URH, WAIT, LEFT, POTTERO, READ, SIGHT, MIDDLE, NO, CHAT, FIRST, WAIT, TBA, PRESS, RIGHT
URH	READY, POTTERO, LEFT, WHAT, CHAT, TBA, SIGHT, NO, PRESS, BLATE, URH, MIDDLE, WAIT, FIRST
LEFT	READY, LEFT, FIRST, NO, MIDDLE, TBA, PLATE, WHAT, URH, WAIT, PRESS, READ, CHAT, MIDDLE, POTTERO
RIGHT	CHAT, SOTHEHO, READY, PRESS, NO, WAIT, PLATE, MIDDLE, LEFT, TBA, PLATE, CHAT, PLATE, FIRST
MIDDLE	BLATE, READY, CHAT, WHAT, PLATE, TBA, PLATE, WHAT, URH, WAIT, PLATE, MIDDLE, FIRST, TBA, SIGHT
*WHAT?	MIDDLE, NO, FIRST, TBA, CHAT, POTTERO, WAIT, CHAT, LEFT, PLATE, PRESS, WHAT, WAIT, SIGHT
*URH?	URH, NO, READY, CHAT, TBA, LEFT, PLATE, PRESS, WHAT, WAIT, SOTHEHO, READY, RIGHT, MIDDLE
PRESS	READY, MIDDLE, TBA, READY, PRESS, CHAT, SOTHEHO, URH, BLATE, LEFT, PLATE, WHAT, NO, WAIT
YOU	SURE, YOU ARE, CHAT, TOTRA, SHEET, URH, URH, HOLD, WHAT, YOU, URH, URH, LITER, DOME, U
YOU ARE	YOU, SHEET, LITER, URH, WHAT, DOME, URH, HOLD, DOME, U, YOU'RE, SURE, URH, YOU ARE
YOU'R	URH, YOU ARE, URH, YOU'RE, PRESS, URH, SURE, U, YOU'RE, YOU, WHAT, HOLD, LITER, DOME
YOU'RE	DOME, U, YOU'RE, URH, SURE, PRESS, URH, DOME, URH, SURE, URH, DOME, LITER, HOLD, HOLD
UR	URH, URH, SURE, PRESS, URH, TOTRA, HOLD, TOTRA, LITER, PRESS, URH, YOU ARE, DOME
URH UR	URH, URH, TOTRA, YOU ARE, YOU, DOME, HOLD, URH, URH, PRESS, URH, LITER, TOTRA, URH, U, WHAT
*URH?	UR, U, YOU ARE, YOU, PRESS, URH, DOME, YOU, URH, LITER, TOTRA, YOU, URH, HOLD, WHAT
WHAT?	YOU, HOLD, TOTRA, URH, U, DOME, URH, LITER, YOU ARE, THE KITE, URH, PRESS, WHAT, SURE
DOME	URH, URH, PRESS, URH, TOTRA, URH, DOME, URH, LITER, TOTRA, URH, LITER, TOTRA, URH, DOME
NEXT	PRESS, URH, URH, URH, DOME, URH, SURE, WHAT, TOTRA, PRESS, HOLD, URH, URH, DOME, LITER
HOLD	YOU ARE, U, DOME, URH, URH
SURE	YOU ARE, DOME, LITER, TOTRA, YOU, HOLD, URH, URH, URH, URH, URH, URH, URH, URH, URH, URH
LITER	YOU, PRESS, U, URH, HOLD, DOME, URH, URH, WHAT, URH, URH, URH, URH, URH, URH, URH, URH

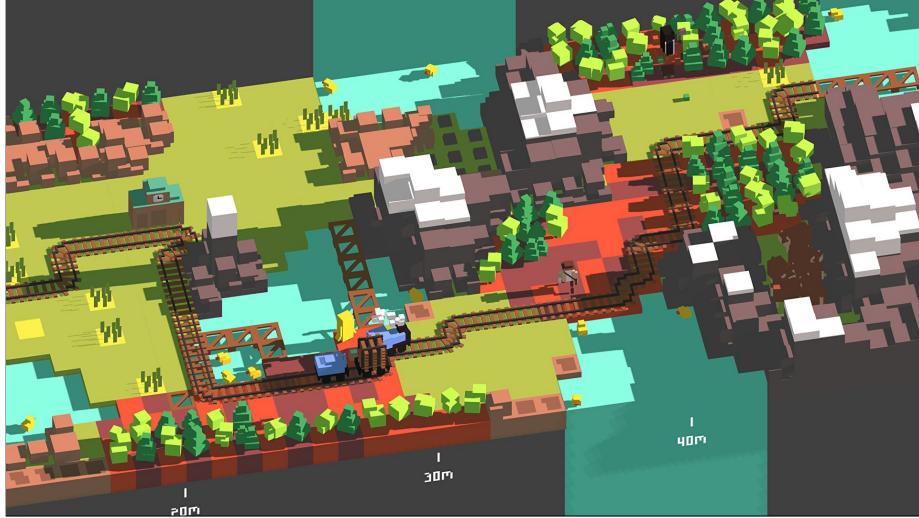
Page 9 of 25

Page 10 of 25

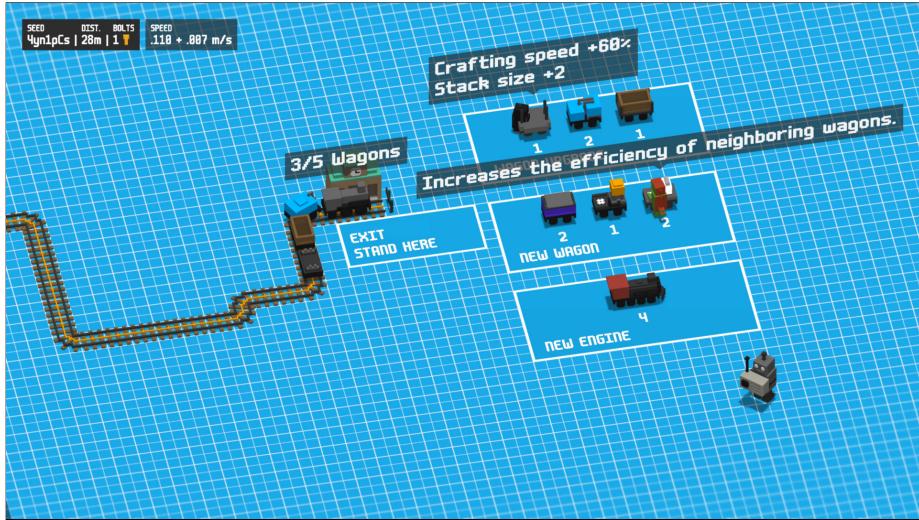
The manual of the game is overcomplicated and explicitly designed to be convoluted. It contains the solutions to the puzzles, but they are obfuscated. Some players have simplified it and memorized it, so they can play without an assistant.

5

Unrailed [7] The game consists in constructing a railway in front of a moving train. If the trains runs out of track, the game is over. In order to construct rails, the players have to chop trees, mine stone and combine materials in a specialized train cart. It's a 2 to 4 player game. The players usually talk to split tasks among themselves and get time-sensitive tasks done quickly.



The game includes pickups, scattered in the map and difficult to get. This pickups are used to upgrade the train and wagons. The upgrade screen is cooperative too, as the players can choose the upgrades by physically grabbing them.



Among Us [1] This game is a mafia game between two teams, impostors and crewmates, which has become recently popular. A team wins if the other team is dead, crewmates also win when they complete all tasks. Impostors have to kill crewmates. If a crewmate finds a corpse, he may call a vote. The player who has to most votes is killed. The players have to discuss who to kill in a limited time.



A player once a game can call an emergency meeting, which triggers the vote without the need to report a corpse. The discussion has to be concise, as there is limited time to convince the other players that they are innocent or accuse another player.



3 Game Design

The game is set in a city, with manhattan grid roads.

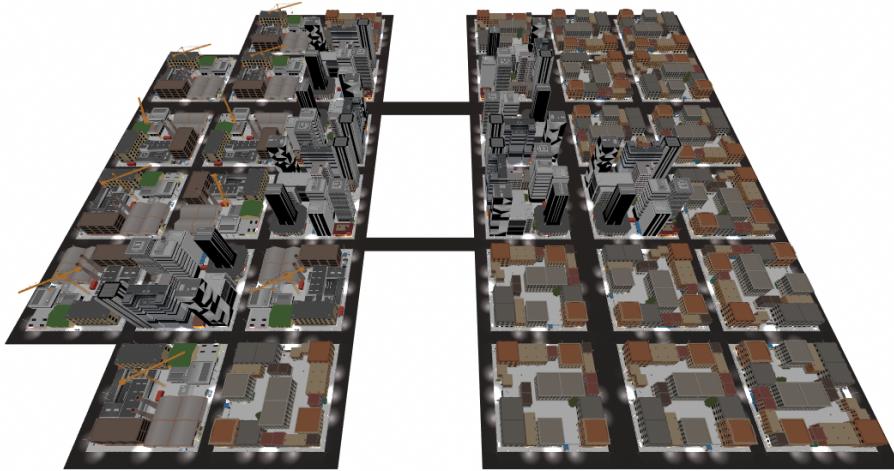


Figure 1: One of the game's maps: Town

There are up to 2 couriers, which have to fulfil a list of delivery orders. There is a master player who has a map, which shows the courier's positions and the street's traffic status. The master has the order list, the couriers do not. The orders consist of a pickup site and dropoff site. The streets are blocked by many obstacles:

- variable level of traffic makes the street difficult to navigate
- a drawbridge over a river
- sidewalk decorations

The master has to plan the route in realtime and relay the path to each courier in the broadcast channel based on the level of traffic. The couriers drive following the Master path, they only see the pickup points and if they have picked up an item, the corrisponding dropoff point. The game is lost if the players did not complete all tasks within the time limit.

3.1 Traffic simulation

The cars are agents that move thorough a graph representation of the streets. The cars roam through the city by turning randomly at intersections. The cars respect traffic lights and occasionally will crash so that traffic jams happen dynamically.

3.2 Items

The items to be delivered are named and have two attributes: lives and weight. When the player has an item and crashes into a wall, the item is damaged. If the item is too damaged, the task is failed and the item reverted to the start. The weight of an item slows the player down by applying an acceleration penalty and a max speed penalty. The weight is not displayed in the game but it's effect are apparent to the couriers.

Item table

Name	Lives	Weigth
water bottle	10	0.9
atomic bomb	4	0.3
money bag	12	0.7
cat	9	1
glass panel	3	0.5
hamburger	6	0.8
box of fireworks	2	0.4
apple	7	1
pizza	10	0.6
chandelier	3	0.4
bowling ball	15	0.1
candy	8	1
pretzel	10	1
bycicle	12	0.4
saxophone	6	0.8

3.3 Master's Information

The information that the master has of the players can be reduced in the settings. The master has three channels of information: geolocation, video feed and audio from the radio. Turning off any of these channels the game becomes significantly harder, except when turning off the radio. The radio is essential for completing the game, so it cannot be switched off. A chatting system may substitute it partially.

3.4 Interface design and menus

The interface has been prototyped directly in Unity. The game is divided in the main menu and the game scenes.



Figure 2: The main submenu

In the main submenu the player can choose a name and navigate to the lobby, settings and join submenu.¹



Figure 3: The settings submenu

¹Bycicle model from Tidominer, <https://sketchfab.com/tidominator>

The settings allows to control the game window, the game's graphic quality and the audio volumes. The menu is constructed dynamically and is easily expandable to accomodate for more options.



Figure 4: The lobby submenu

The lobby is where the master (which is the host) accepts connections and configures the game. The players can see the configuration and chat vocally and each other.

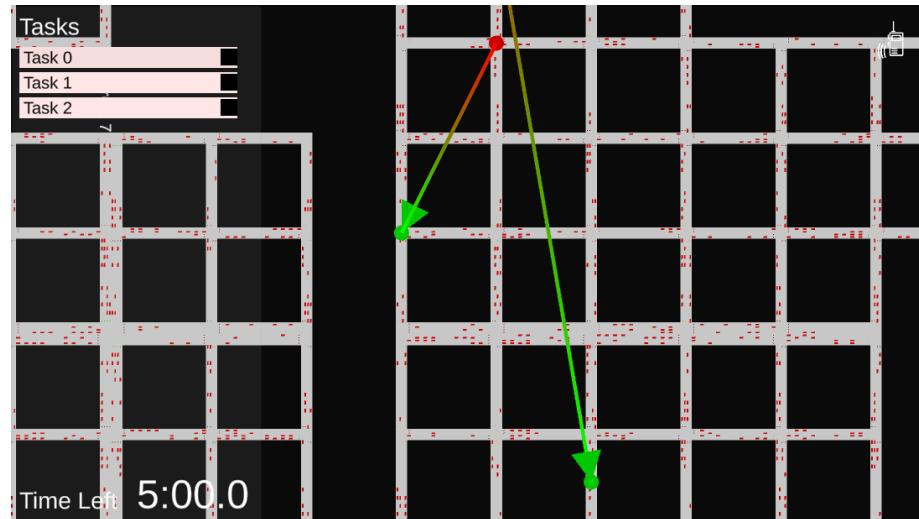


Figure 5: The master interface

The master has at his left the task list and the time left in the game. In the bottom right there are the video feeds of the players and an arrow connects the feed to the corresponding player. The arrows which connects the pickup place and the dropoff of a task can be toggled by clicking on the corresponding task in the list. The task status (none, worked on, completed) changes the color of the task in the list.



Figure 6: The player interface

The player can see the carried item name and the amount of lives remaining until the item is broken. When a player loses a life an animation plays flashing a white vignette and decrementing the lives counter.

4 Implementation

4.1 Level design and 3D models

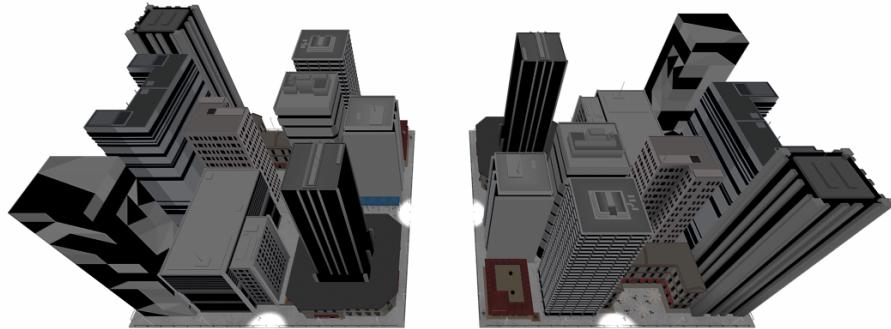


Figure 7: Downtown block

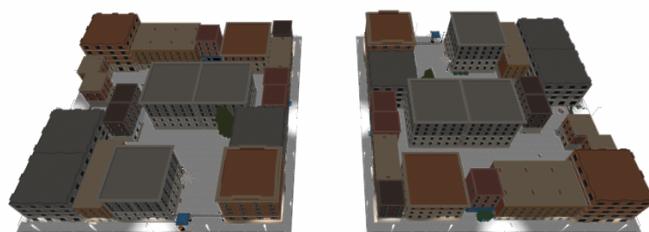


Figure 8: Residential block

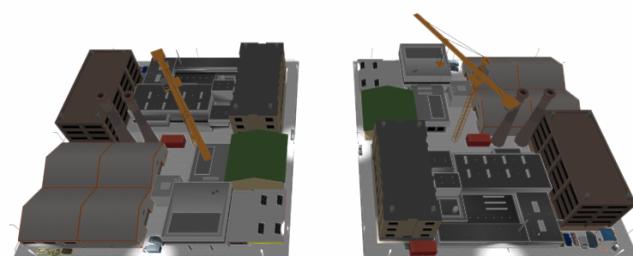


Figure 9: Industrial block

The game maps are constructed by placing blocks, which are made of multiple buildings. Blocks are all contained in a square with a 110m edge. Blocks are assembled using the Everything Library². The roads are placed between blocks and their width is reflected in the number of lanes. Signs that indicate the names of the roads are generated following the road layout. Traffic lights and cars are generated at runtime. This workflow allows for big maps to be constructed quickly. In the game there are three maps: City, Town and Village. The maps are stored in separate scenes, so changes can be affecting only one map. This allows to have different lighting, decorative objects and gameplay elements.

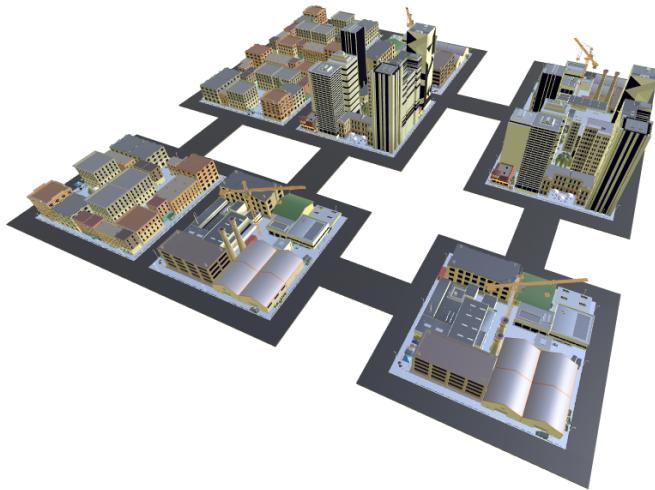


Figure 10: Game map: Village. This map is the smallest and has the most bridges

²Assets from Everything Library © David OReilly, <https://www.davidoreilly.com/library>

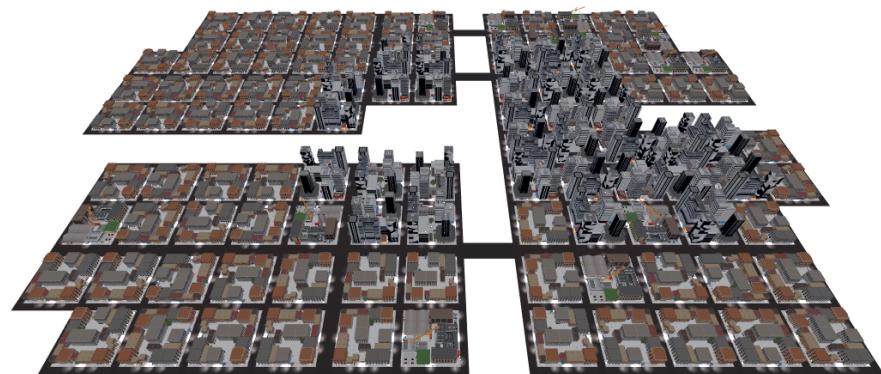


Figure 11: Game map: City

4.2 UDP Sockets with fragmentation and optional retransmission

The connection is handled using native C# asynchronous UDP sockets. The message is sent with a protocol number, a serial identifier that identifies the packet and an id string that identifies the sender.

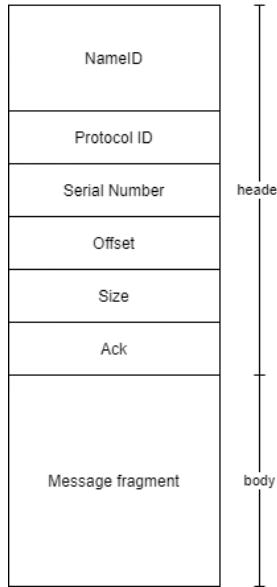


Figure 12: Network packet composition

Fragmentation If the message is longer than 1024 bytes, it's fragmented in parts no longer than 1024. Each part has the protocol, the serial number and the id along with an offset and a size. The offset is what part of the original is sent and the size is the size of the whole message. At the receiving end, the datagrams received are assembled back into the original messages using the serial number, the offset and size. The partial messages are stored in a dictionary indexed by the serial number and are made available externally only when full. If the fragments arrive not in order it isn't a problem as they are assembled based on their offset.

Optional Retransmission If the packet is marked as resend the system will wait for a response (ack) indicating that the packet was received. The system checks periodically if the timeouts of each packet is over. If the ack isn't received and the timeout is over the system will resend the packet with the same serial number.

4.3 Game Architecture

The game is based on the model-view-controller pattern. [9]

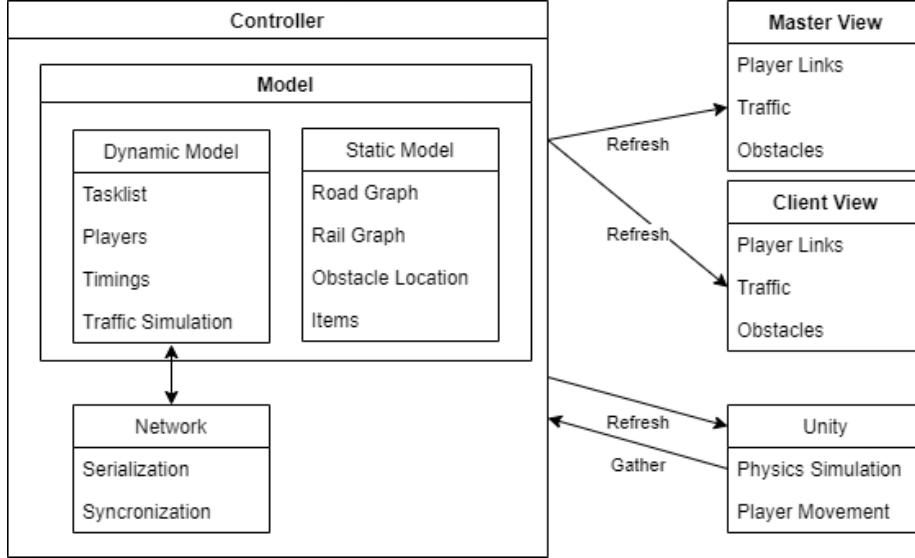


Figure 13: Game architecture diagram

Model A subset of the game rules which constitute the model are implemented decoupling the code from Unity. This subset is everything apart from the physics simulation, player movement and input management. The reason behind the separation from Unity is that the master mode has to be as light as possible. The master mode is meant to be able to run smoothly on slower devices and be ready to be ported to a native browser integration, which is much harder if it relies on some engine features. The model is divided into static and dynamic. The dynamic model is the state that changes during the game and is synchronized from the master to every client. The static model is loaded at the start and contains unchanging data.

View The view is split into master and client. In both instances, it creates a visual representation of the model and links each element to the model with its identifier. When the model is changed, the view is updated.

Controller The controller interfaces with the model, the view, the networking and Unity. The controller behaves differently between client and master. The client gathers player data from and sends it to the master through the network. The master applies the client changes and sends back the updated dynamic model state.

4.4 Synchronization

Multiple instances of a game can be synchronized either by passing only the inputs or the whole state.

Inputs synchronization By passing only the inputs, every instance has to update their simulation based on all inputs gathered before the previous update. As the master doesn't have a full simulation of the game (physics and player movement are only in the clients), this approach is unfeasible.

State synchronization The master sends the whole dynamic state to the clients. The clients pass the outcome of the inputs in the form of the player position and a flag if the player happens to collide with the environment. This approach is unfeasible if the state is too large to be efficiently sent tens of times a second over the network, but in this game's case the dynamic state is small enough.

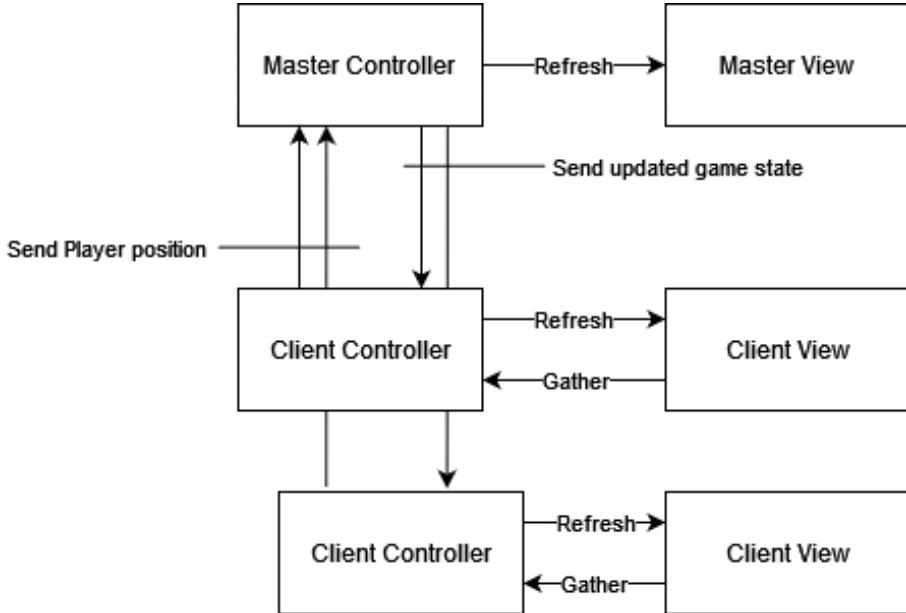


Figure 14: Network Architecture diagram

4.4.1 Dead reckoning and latency hiding

The network layer is unreliable and has variable latency. To mitigate for the delay and the unreliability there are multiple solutions: introduce a fixed delay or rollback.

Delay The fixed delay (usually less than 100 milliseconds) negates the variance in latency if the packets all arrive before the fixed delay. If one packet is late, every instance of the game must wait for the missing packet, ignore it and move forward without or rollback when it arrives.

Rollback When the server receives a late packet, it reverts the game state to the time it was sent and reupdates it back to the current time. While waiting for the late packet, the server computes the next states trying to predict the missing packet effects. This extrapolation, also called dead reckoning, can be effective with a trivial implementation when using input synchronization (use the inputs from the previous packet). The game uses state synchronization, so the extrapolation is nontrivial to define.

Extrapolation The extrapolation used in the game involves only the remote players position in the client view. The position is modified by the speed of the players based on how much time has passed since the last packet and the last speed known is kept constant.

4.4.2 Advantages over HLAPI [6]

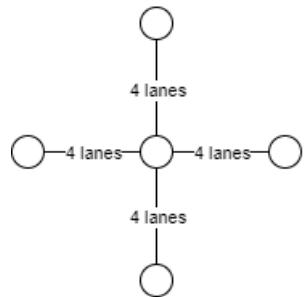
Unity provides the multiplayer high level API which is a system that offers messages, serialization, state synchronization and network classes. HLAPI is deprecated and highly integrated in Unity, which is not desireable as it makes it very difficult to decouple the state of the game from Unity.

4.5 Agent Based Traffic Simulation

4.5.1 Definition and Assumptions

The agents of the simulation, cars, are defined by their position relative to a graph, a velocity and other parameters (random acceleration, random seed). This graph called "rail graph" is derived from the street level graph, which is a graph that defines the layout of the roads. The rail graph gets its name from the cars behaving more like trains or trams, as they can't deviate from the rail graph edges. The rail graph is constructed by replacing every node of the road graph with an intersection and every edge with lanes. The road graph has a parameter for every edge indicating how many lanes the street has.

Road Graph



Rail Graph

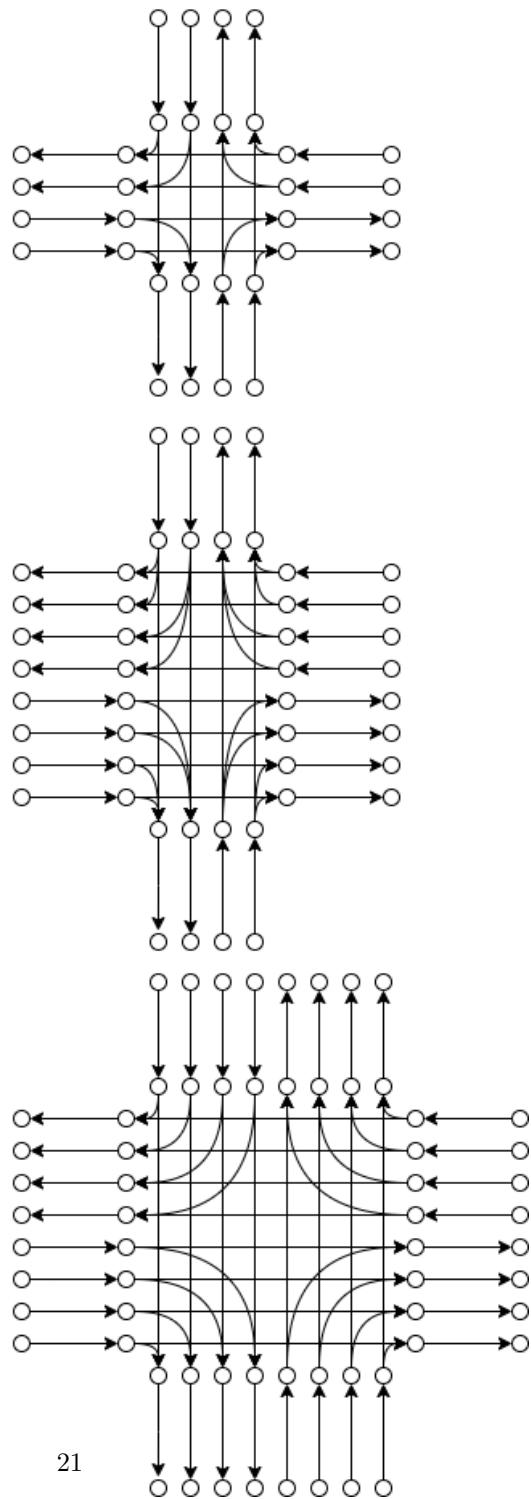


Figure 15: Rail Graph construction from the Road Graph

The rail graph is directed and the direction of an edge is the direction of the flow of traffic. At the intersection the rails are connected by bezier curves. The left turn rails are not connected to not allow cars to turn left, as it causes a jam instantly.

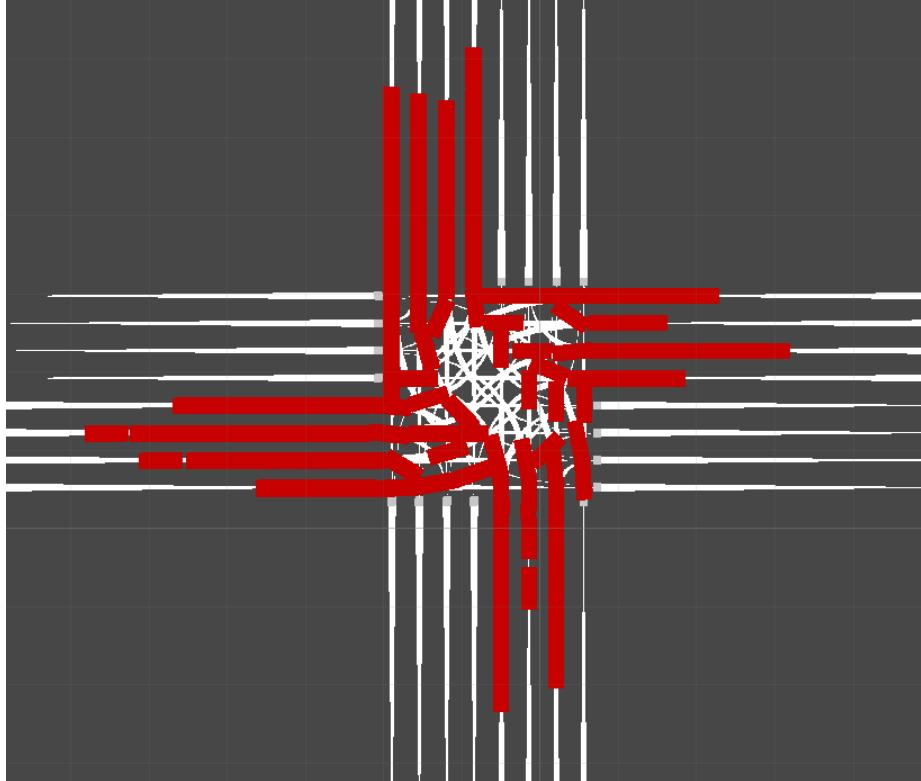


Figure 16: Left turn without traffic lights causes instant jam, this is the state after only 10 steps

To alleviate this jam, the left turning rails would have to be non intersecting and a left turn semaphore would have to be put in place. However, it is not necessary to do so as every point is still reachable without turning left.

Traffic Lights Traffic lights are placed on the intersection's inward rails. These lights are coordinated at a street node level, so every traffic light in an intersection is controlled by a single timer. Based on the position of this timer, the traffic light's state is set. The traffic lights are grouped into odd and even parity: odd parity lights are phase shifted by half a timer cycle. This grouping makes lights switch on and off based on their direction.

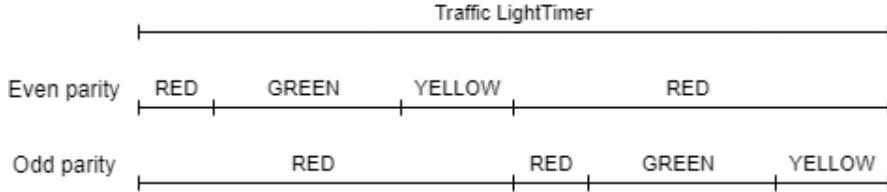


Figure 17: Traffic light timer and states

4.5.2 Implementation

Car-Train Agent The car's position is defined by the starting node, the ending node (so the edge on which the car is) and the relative position $rel \in [0, 1]$ along the edge. The absolute position of a car is the position of the car within the frame of reference of the position of the nodes of the rail graph.

To get the absolute position a linear interpolation based on the relative position from the start to the end node is performed. If the edge on which the car is on is a bezier, the position is just the evaluation of the equation at the car's relative position.

To get the absolute direction the normalized difference of the end and start node is sufficient for the linear case. In the bezier case, the derivative of the bezier equation is the tangent vector, which normalized gives the absolute direction.

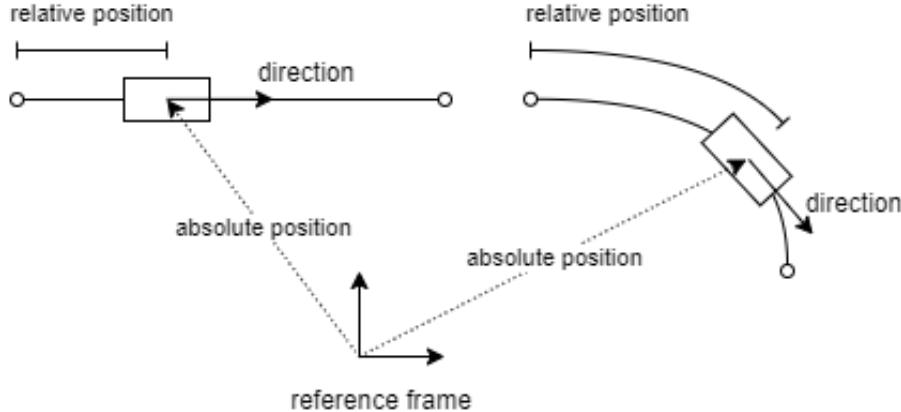


Figure 18: From the relative position of a car to the absolute position and direction. Left: linear, Right: quadratic bezier

Integration The traffic state is a collection of cars. The state is integrated discretely with a time differential dt . Every car calculates how much distance it has to move by using its velocity. This distance is then consumed by navigating the graph. The car's relative position is incremented by the relative distance, which is the distance divided by the edge length. This is easily calculated

for linear edges, but bezier edges have a complex elliptical integral to solve to get their length [2]. Therefore, the relative distance is calculated using an approximation of the bezier length by sampling a few points.

When a car reaches an intersection, if it has a red semaphore it stops. Otherwise, it chooses a rail based on its initial random seed, which doesn't compromise the deterministic property of the simulation.

4.5.3 Optimization

Collision Detection: Lookahead dictionary Before the cars are moved, a lookup dictionary is constructed. This lookahead dictionary stores for every car the position it would have had if it moved some integration steps. The number of lookahead steps is a sensitive parameter with respect to number of incidents and runtime speed. Every time a car is moved, it checks if it would intersect any other future cars using the lookahead dictionary. If it would intersect, the car is stopped.

The intersection between two cars is an overlap check on two circles per car that approximate the bounds of a car.

Lookup optimization and caching Every collection used in the simulation can be accessed randomly in constant time and the absolute position and direction of a car are recalculated only when the relative position is modified. The rail graph has precalculated structures such as an index for accessing edges directly and a forward star collection for every node.

Intersection optimization: Grid indexing The intersection checking is the most costly operation for the simulation. In order to check faster, the cars further than a radius are excluded from the check. This is a coarse filter, but it requires a quadratic number of checks w.r.t. the number of cars. Therefore, an index based on the car's position is implemented. This index divides the space in a grid and for every square it has a collection listing all cars contained in the square. This index can be queried to return a neighbor of a car, and it does so by returning all cars in the original car square and all adjacent squares.

Parallel Integration The integration and collision detection algorithm can operate on a copy of the traffic state (which is a dictionary of cars) and apply the changes at the end of computation thanks to the lookahead dictionary. This makes it possible to parallelize the creation of the lookahead dictionary, the position integration and the collision detection. The threads have access to all the indexed data and traffic state and place their result into thread-safe containers. The threads work on a subset of cars and can't contaminate each other subset as they are changed at the end of computation.

Stopped cars linking The cars which are traveling and detect a still car on their path are stopped and linked to the other car. This link is broken when

the car detects that the linked car is moving. When a car is linked to another, it's not needed in the lookahead dictionary and can skip being moved.

This linking procedure helps speeding up the intersection check, which is the slowest part of the whole simulation. Although, in the initial state the cars are placed in such a way that no car is linked, which forces the simulation to run slower until some cars are stopped in order for the linking procedure to take effect.

4.5.4 Traffic Synchronization

The traffic simulation is synchronized by passing the current step of the master to the clients. The clients will advance their simulations until they reach the same step as the master, then wait until the master step is incremented. The initial state of the traffic model, which is composed of the cars and the traffic lights state, is sent from the master to the clients at the start of the game.

4.6 Kinematic Bycicle Model [10]

The model is a simplified car steering model, where the front and back wheels are collapsed into one front wheel and one back wheel. The only steering wheel is the front one.

Known constants:

$$l_r = \text{distance from the center to the rear wheel}$$

$$l_f = \text{distance from the center to the front wheel}$$

State variables:

$[x, y]$ = absolute position of the bycicle in the plane.

v = velocity of the bicicle.

ψ = heading angle

Control variables:

u_1 = acceleration

u_2 = steering angle of the front wheel.

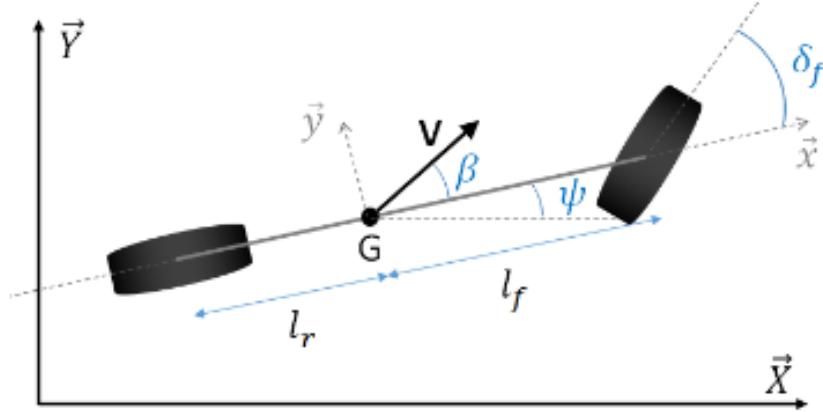


Figure 19: Kinematic bicycle model

The model's differential equations:

$$\dot{x} = v \cos(\psi + \beta(u_2)) \quad (1)$$

$$\dot{y} = v \sin(\psi + \beta(u_2)) \quad (2)$$

$$\dot{v} = u_1 \quad (3)$$

$$\dot{\psi} = \frac{v}{l_r} \sin(\beta(u_2)) \quad (4)$$

where $\beta(u_2)$ is the slip angle given by

$$\beta(u_2) = \arctan\left(\tan(u_2) \frac{l_r}{l_f + l_r}\right) \quad (5)$$

The model is integrated discretely.

$$x_{t+1} = x_t + \dot{x}\Delta t \quad (6)$$

$$y_{t+1} = y_t + \dot{y}\Delta t \quad (7)$$

$$\psi_{t+1} = \psi_t + \dot{\psi}\Delta t \quad (8)$$

4.6.1 Player movement

The player is moved following the KBM model. The Unity vertical and horizontal input axis (which contain the standard movement input: arrows, wasd keys and gamepad sticks) are assigned to the control variables.

4.7 Radio

The radio in the game is a simulation of a real half-duplex radio. This means that only one player at the time may speak. This is enforced by transmitting white noise while two or more players are using the radio channel simultaneously. The simulation relies on a server based network to centralize the audio mixing process. All clients send their audio to the server, which mixes all the sources and sends customized mixes to each client and the local speaker. As a convention, all audio passing through the mixer has a sampling rate of 48kHz.

Voice Loopback The client audio doesn't loop back to the sender. When the voice returns to the speaker it's delayed, causing confusion. The server processes all audio sources to calculate if noise is needed, then for every client it mixes all sources except the client one.

Buffers The client audio is sent to the server and cached in a buffer in order to account for desynchronizations and network jitters. The server creates and fills a buffer for every client. Periodically, the server polls the buffers and mixes the polled data. The mix is sent to each client where it is cached in a buffer. This buffer is constantly polled by an Unity audio callback. The draining and filling of the buffers is balanced: the microphones produce as many samples as the ones consumed by the speakers, so the intermediate buffers serve as a transfer node.

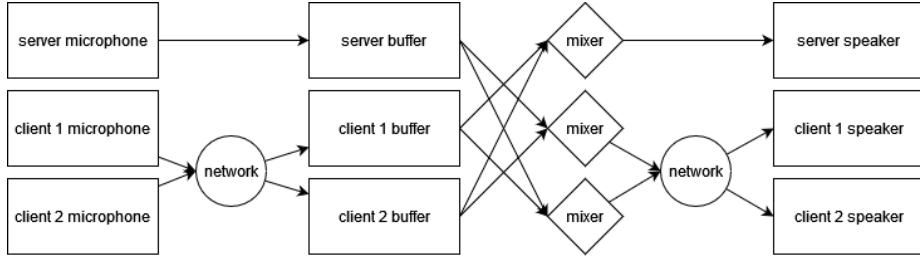


Figure 20: Audio pipeline diagram, the boxes represent the buffers.

Buffer delay Every buffer delays the signal by a set amount. This delay is crucial to alleviate the network variance, which otherwise would cut the signal and override samples. The buffers, when queried to read a certain amount of data, return silence if the data available is less than the requested amount. This ensures that the mixer will always have either a full set of data or silence and that the buffers will not cut the audio.

Sampling frequency and Resampling The client's microphone may have a different sampling frequency from each other. The audio sent is resampled to 48 kHz and the audio received is resampled to the speakers sampling frequency. The sampler used is a custom sample and hold algorithm, it inserts or deletes samples following the ratio of sample rates.

Noise and Filters [11] The noise generated and mixed when multiple clients are speaking is white noise at 48 kHz. The audio generated by the clients is filtered by the server with a custom hi pass filter, which attenuates the low frequencies. This effect is done to make the simulation closer to an old style radio.

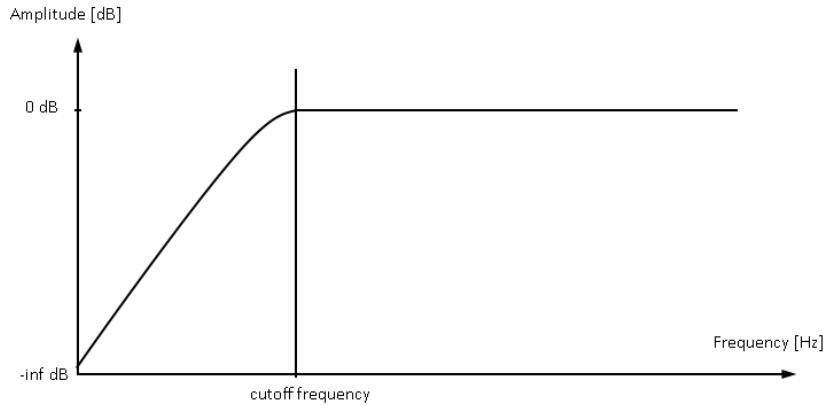


Figure 21: Hi-Pass filter Bode diagram

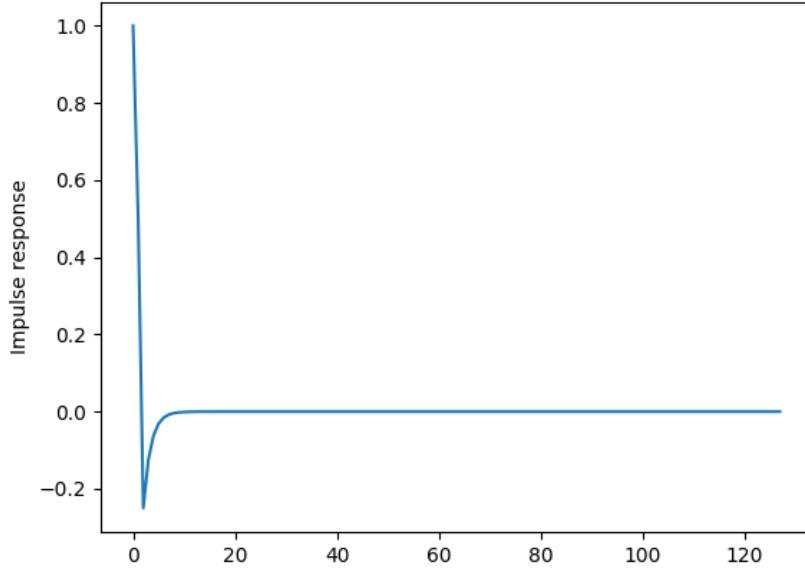


Figure 22: Hi-Pass filter Impulse response

The filter is a recursive digital filter, defined by the following differential equation:

$$y(n) = \alpha(y(n-1) + x(n) - x(n-1)) \quad (9)$$

4.8 Video

The video feed is a frame of the player's camera which is sent to the master by each player. The amount of data required scales rapidly with the resolution of the frame and the frequency of the transmission.

4.9 Editor Tools

The development process was sped up by the construction of tools for Unity. This tools are based on editor scripts, which allows to automatize some actions in the editors such as instantiating prefabs (keeping the prefab link) and destroying GameObjects.

Map Generator The map generator places prefabs on a grid, randomizes their z rotation snapping on the 4 cardinal rotations and swaps these prefabs with their updated ones. This script also places road segments ensuring that no segment overlaps another and giving each segment a unique identifier.

Graph Generator This tool generates a graph (a nodes and arcs tuple) from the elements present in the scene. The graph is saved directly in the Assets/Resources folder in json format. The graph can be visualized in the scene with a toggle in the script. The tool also visualizes the traffic rail graph, the cars and the traffic lights state. The traffic simulation can be stepped through directly in the unity editor, which eliminates the need to wait for unity to reset it's state while entering play mode.

5 Testing

5.1 Unit Testing

Unit tests are automated tests, they ensure that a small section of code behaves as intended. It's difficult to write testable code and to write tests for all the required functionality if the tests are not included in the design. To make it easier to write a complete test suite (set of tests of a program), a technique called test driven development is used to write code and tests.

5.1.1 Test Driven Development [8]

The model of the game is developed using the test driven development technique. This technique allows writing testable code by first writing a failing test and then the code to pass it. This makes the code more robust and easier to clean. Other aspect of the game such as the controller, the view or the map tools are not easily testable this way as they depend on the unity framework. One of the goals of TDD is to make it easy to refactor code: if the refactored code passes all tests the code is correct, there's no need to run it to know. This relies on the robustness of the test suite. If an essential functionality is not tested, the test suite is incomplete and by running all tests there is no guarantee of correctness.

5.2 Integration Testing

The integration testing phase is largely done by hand. This is necessary as unity does't have a way to run multiple deployed instances inside a test environment. The testing routine is to deploy an executable and run it parallel to the editor instance. Some components are tested automatically, such as the serialization of the model and the interaction between smaller model components.

5.3 Clumsy [3]

Clumsy is a program that makes the internet network worse. It simulates packet loss, jitter, latency, bandwith problems, it tampers with packets. This tool is the basis on the game's retransmission algorithm, because it creates the perfect network conditions for testing.

6 Conclusion

Feedback

Further development

References

- [1] Among us. <https://www.innersloth.com/games/among-us/>.
- [2] Bezier info, an online book on bezier curves. [https://pomax.github.io/bezierinfo/1](https://pomax.github.io/bezierinfo/).
- [3] Clumsy source code. <https://github.com/jagt/clumsy>.
- [4] Keep talking and nobody explodes. <https://keepTalkingGame.com/>.
- [5] Keep talking and nobody explodes, the elevator of doom by lthummus. <https://www.youtube.com/watch?v=y9zGRy1bl1A>.
- [6] Unity high level api documentation. <https://docs.unity3d.com/Manual/UNetUsingHLAPI.html>.
- [7] Unrailed. <https://unrailed-game.com/>.
- [8] Kent Beck. *Test-Driven Development: By Example*. Addison-Wesley, 2002.
- [9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.
- [10] Philip Polack, Florent Altché, Brigitte Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? pages 812–818, 06 2017.
- [11] Julius O. Smith. *Introduction to Digital Filters with Audio Applications*. W3K Publishing, 2007.