Shadow Spy
Introduction to Computer Graphics (COS426)

Simon Sure (ss9971@princeton.edu / simon.sure@inf.ethz.ch)
Flurin Steck (fs7278@princeton.edu / flurin.steck@inf.ethz.ch)

December 12, 2024

Shadow Spy is a two-player hide-and-seek game. Each player navigates a dark shared world from his own first-person perspective carrying flashlight. A player wins by finding their opponent and observe him them sufficiently long first. Players can choose between different scenes.

The game is implemented with `three.js`, decentralized WebRTC based browser-to-browser communication, and Blender 3D models with animations. We make extensive use of libraries for audio, raycasting and other features, and custom implementations collision detection and handling, first-person controls, and more.

Contents

1 Goals **2**
1.1 Networking . . . 2
1.2 Graphics Software . . . 2
1.3 Functionality . . . 3

2 Execution **3**
2.1 First Person Controls . . . 3
2.2 Game State Machine and global state . . . 3
2.3 Networking . . . 4
2.4 Software Architecture . . . 4
2.5 Minimap . . . 4
2.6 Score Keeping . . . 4
2.7 Player Model & Animations . . . 4
2.8 Scenes & Scene Selection . . . 4
2.9 Audio . . . 4

3 Results **4**

4 Discussion **5**
4.1 Pairing & Scaling 5
4.2 Customization 6
4.3 Lighting 6

5 Ethical Evaluation **6**

6 Resources **6**

7 Contribution **6**

1 Goals

Looking at past projects we decided to bring a new and unique feature to our game. We decided to make a multi-player game which influenced all subsequent design decisions.

The idea for the gameplay was quickly developed. The *Results* section elaborates extensively how the game works. We discuss more specific goals in this section.

1.1 Networking

Implementing a multi-device multiplayer game requires networked communication between the players. Most games use a centralized setup. To (a) not having to operate the server infrastructure and (b) not having to develop a separate server backend, we wanted to utilize direct browser-to-browser communication. This technology has been advanced in recent years, mostly as part of WebRTC.

Our plans for a multi-player game would require synchronization. Because the goal was to randomize the scenes, we couldn't only load static resources. During the game, the player movements would have to be synchronized at a sufficient rate for an interactive gameplay.

Using direct browser-to-browser communication generally provides lower latency than a centralized approach. This makes the game even more interactive. When both players are physically close, the latency is practically unnoticeable.

1.2 Graphics Software

We knew from the beginning that our game would require a lot of functionality to support a smooth user experience. We would have to implement first-person controls, use ray-casting to detect when one views another user, perform collision detection, support for multiple scenes, support for networking, ... Thus we put a focus on a modular high-level software architecture for our project.

1.3 Functionality

Our initial ideas for additional features went far beyond what we had time to implement. We discussed adding weather, placing power-up items in the world, ... Given careful planing of the software architecture, those features would be reasonable to implement with a considerable time investment.

Considering the time limitation, we created a list of essential features for the game to be usable and focussed on implementing those quickly. Afterwards, we could add features as time allows.

The essential features were: First-person controls, scene bounds, score keeping, mini-map, a basic static player model, and network sync. But to have a not only technically playable but practically usable game, we had a set of very important features: Nice scenes, collision detection, and proper player model with animations.

We managed to implement all essential and utmost important features but not much more in the available time. More on additional features in the *Discussion* section.

2 Execution

We discuss the development of the project in chronological order. This section focusses on the technical aspects of the project. We discuss the non-technical aspects in sections *Goals* and *Results*.

2.1 First Person Controls

walking: keypress handler for wsad keys,

viewing: pointer lock

scene hierarchy initial: player in scene, camera attached to player, rotate/move entire player coordinate frame, determine forward/right vectors of current camera view, use those for walking offset, determine camera x (right) and y (up) axis in global terms to do rotations

walking with right/forward done; reposition entire player, rotation around y also on entire player, rotation around x on newly introduced head to which camera attached (camera fixed on rotation, order of rotation easy; not quaternion mess)

2.2 Game State Machine and global state

maintain state machine for different phases (game start, connecting players, network setup/sync, scene selection and scene generation, gameplay, determining winner and winning screen

global state of important data such as whether player A or player B, scene, network status, ...

2.3 Networking

not rely on custom server architecture, direct P2P communication requires considering consistency and parallelism challenges when maintaining game state

using WebRTC (web real time communication; also used by some video conferencing services), best supported in Chromium-based, also Safari; issues with Firefox

use public pairing server for discovery of other player; afterward direct browser to browser communication without centralized server

2.4 Software Architecture

2.5 Minimap

2.6 Score Keeping

2.7 Player Model & Animations

2.8 Scenes & Scene Selection

2.9 Audio

3 Results

The final game provides a simple user experience. The game can be played by running the java script application locally or visiting https://roldsi.github.io/shadow-spy/. It is also possible for one player to use the locally deployed version and the other player to use the web-version.

To initiate a game, one player 'Player A' and the other player chooses 'Player B' on the splashscreen. Player A enters the code shown to player B. This establishes the connection between both browsers. Player A can then configure the game by choosing a scene.

- `BaseScene` is a simple plane without any objects and was originally introduced for development purposes.
- `FlowerHorror` is a flat plane that contains a few flowers at randomized positions.
- `Terrain` is a complex scene. It contains a randomized elevation profile, randomized distribution of objects such as trees, stones, etc.

As soon as player A choses a scene the game starts for both players. The user is placed at a random position in the world sufficiently far away from the other player. Depending on the browser, it may be necessary to click onto the browser window once to activate first-person controls. One can walk with keys 'wsad' and rotate by moving the cursor.

While walking the world, players can't walk through objects. Further, the light source of the flashlight is linked to the flashlight of the 3D player model. Thus, the light beam will move while walking. The flashlight has a limited range and angle. There is also background music.

On the bottom right, the player can see a mini-map of the world with its own position. Above, one can see one's own count and the other player's count. One gains points whenever one sees one's opponent. Peeking from behind an object at the opponent only counts when a sufficiently clear line of sight exists.

If one is observed by the opponent, one doesn't see the opponents score increase immediately. Their gains will only be shown once one is no longer observed to avoid using the score counter as an indicator of the presence of the opponent.

If both players observe each other, they are repositioned to new random locations in the map to avoid a stalemate.

The first player to reach 1000 points the game. As soon as that happens, the game is terminated for both players and notified whether they have won or lost. One can immediately start a new round.

4 Discussion

Any project can be improved arbitrarily. At this point, we are not aware of any big issues or bugs with the game but have various ideas for additional features or possible improvements.

We only list a selection of our ideas here to keep the report reasonably short.

4.1 Pairing & Scaling

Initially, the pairing process required that player A enters a long randomized string provided by the pairing server because every active connection with the pairing server must have a unique id. We avoid manually typing a long random string, we generate a short player code and appending a long game-specific string. This string is likely to be unique but that is not guaranteed.

If the game would be played by hundreds of people simultaneously, it will be likely that the same ID is used by multiple players. The game synchronization will break in that case.

A solution would be a more advanced pairing and discovery protocol. Considering that his is a graphics project at the core, this was out of scope.

4.2 Customization

Player A can choose from a limited set of scenes at the beginning of the game. One could add more different scenes and improve the degree of randomization within those scenes.

Additionally, one could provide the user with more customization options such as different characters to choose from.

4.3 Lighting

Given the pre-defined player model with its flashlight, there is currently limited variety in scene lighting. The game could become more interesting by adding choices between different types of flashlights: Long-range but narrow, very wide but short-range, ... Having a brighter flashlight is not necessarily an advantage because one can be tracked easier.

A great feature would also be to allow players to turn off their own flashlight. This increases the game complexity because one can try to use the other player's light to stay undetected oneself.

Besides basic flashlights, one could also introduce power-ups that can be gained throughout the game. For instance, a player could gain night vision for a limited time. This would give a one-sided advantage.

An even other option to introduce variety would be weather. Rain and fog could make visibility more difficult. While lightning could add bright momentary global illumination.

5 Ethical Evaluation

6 Resources

- related work

7 Contribution

by group members