Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed
Computing

Undercover Construction: A Multiplayer Game with Hidden Aspects

Semester Project

Philipp Friedli

phfriedl@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger
Prof. Dr. Roger Wattenhofer

June 12, 2018

Acknowledgements

I would like to thank Manuel Eichelberger for his continuous support. His inputs helped in shaping the game to what it is today. Big thanks goes to all the friends who gave honest feedback, as well as the Computer Engineering and Networks Laboratory (TIK) at ETH Zurich for offering this opportunity and to provide the necessary hardware.

Abstract

The main ideas behind *Undercover Construction*, a multiplayer strategic action game, are presented. *Undercover Construction* makes use of asymmetry in information to hide progress from opposing players whilst using a single screen. It uses hidden input and delays to achieve this. The winning condition is to finish building a tower before all opponents.

Contents

Chapter 1

Introduction

It is easy to hide information from opponents when gamers are playing on different screens. When players share a screen, the designer has to decide on what should be displayed. The longer a game lasts, the more details should be shown to support the players' minds.

For example, one could expect a player to remember a single item he is holding. But asking of the player to memorize lots of items, all of which he might only use minutes later in the game, might leave him frustrated. In *Undercover Construction* the goal of each player is to collect items and build a tower without letting the others know as long as possible. Opponents should not be given the opportunity to interfere. If the game directly displays somewhere that a player is picking up a specific item, it might reveal his identity too easily.

Each action a gamer can do, can either be visible or invisible. In a game with hidden aspects, a designer has to carefully weigh the need of displaying something against the usefulness for the hidden gameplay if he does not. Different approaches to this problem are presented.

This work explores ideas in creating a game that mixes luck, strategy and skill, while demanding from the players to hide from each other.

1.1 Related Work

The starting point of this project is *Hidden in Plain Sight*(HiPS)[1], which is a local multi-player[2] video game that consists of multiple mini-games. The main principle is to hide oneself among many *Non-Player Characters (NPC)* by not making any noticeable movements which would reveal the players location.

Our game, *Undercover Construction*, uses NPCs as a hiding mechanism for characters similar to HiPS. In contrast to HiPS, the main goal is not simply to find opponents, but to collect items and build a tower in secret, such that the opponents cannot interfere. Delays and randomness are used to strengthen the aspects of hiding, luck and the need of planning ahead.

[1]https://store.steampowered.com/app/303590/Hidden_in_Plain_Sight/

[2]Local multiplayer: multiple players play in front of a single screen without any network connection [1]

Chapter 2

Game Mechanics

This chapter gives an overview of how the game works and what design decisions were made to shape the gameplay. Section 2.1 describes the inputs the player can give and what rules and game objectives there are. Decisions about which information is hidden during a match are explained in Section 2.2, whereas in Section 2.3 some possible ways to benefit from these mechanisms are explained.

2.1 Gameplay

Wikipedia gives the following definition of *Gameplay*:

> **"Gameplay is the specific way in which players interact with a game, and in particular with video games. Gameplay is the pattern defined through the game rules, connection between player and the game, challenges and overcoming them."** [2]

Undercover Construction is played on a static field which is split into an inner arena and an outer arena, see Figure 2.1. The view is isometric to give an artificial sense of depth [3].

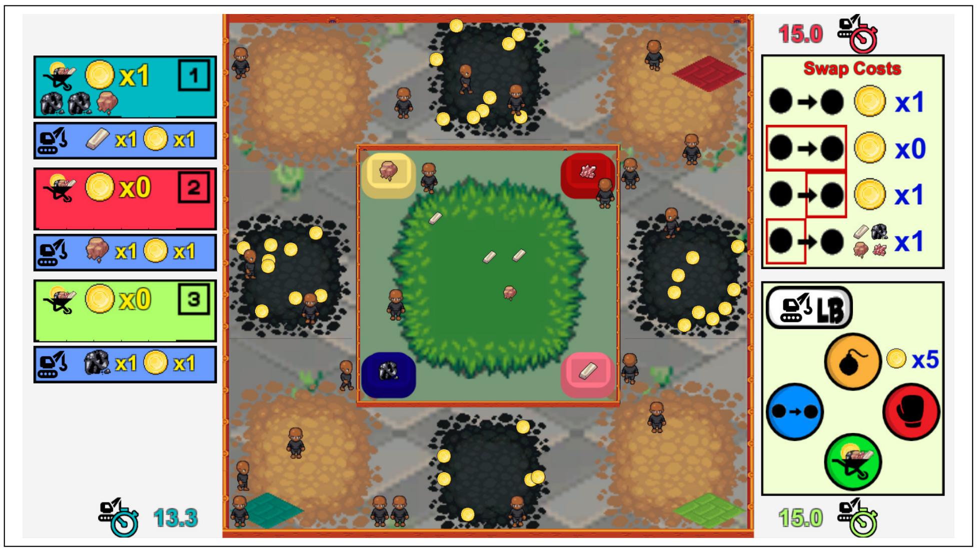


Figure 2.1: Screenshot showing a match of *Undercover Construction* with three active players.

The game is designed for two to four players. In each corner of the field there is a tower assigned to each participating player. In between, there are black fields that generate and host coins. The middle arena consists of four in-game buttons which can be activated by standing on them. The active button determines which item is to be spawned on the green meadow. Multiple identical characters are placed across the playing field. Each player starts of by controlling one of the characters in its own tower's neighborhood. The middle arena hosts exactly as many NPCs as there are active players.

The **goal** of the game is to build four additional levels of the tower to reach level five. The starting level is considered as level one. Each level needs another item and each player requires a different item at the start. The required items rotate for each new level and the amounts required increase by one. To reach level two a single item is required. The costs in terms of coins are calculated with the simple formula:

$$cost = 2 \cdot level - 1,\ level \in \{1, 2, 3, 4\} \tag{2.1}$$

2.1.1 Actions

Besides the joystick for movement, five buttons of the gamepad are used. Each of the following buttons have an action assigned.

• **Button A**	Item interaction and storage
• **Button B**	Attack
• **Button X**	Swap
• **Button Y**	Plant bomb
• **Button LB**	Build tower

The most used action is the interaction with **items**. This action can add an item or coin to the bag when standing over it. The same action is used to store items in the tower. Coins cannot be stored and are always in the bag. A player can hold up to 15 coins and 5 items. The item bag is like a queue with a fixed size. The oldest item in the bag is discarded when adding a sixth item. The storage capacity of the tower is unlimited.

The **attack** action is used less often, but it is a powerful as well as a dangerous action. The action kills another character with one hit. Killing an enemy-controlled character benefits the killer by adding items required for the next level to the item bag. The number of items the player gets is equal to the level his opponent is currently at. By giving a stronger motive to look for players that are closer to winning, the players that are behind get a chance to catch up. Players get punished when they kill an NPC. The punishment is the loss of all coins, which is why attacking random characters is not beneficial. A killed player takes control of a random character in the outside area after recovering.

An important action in connection with hidden information is **swapping**. This action allows the player to swap places with any NPC. Swapping is limited to horizontal and vertical directions. It requires one coin to swap per default, except when the player needs to swap out of the middle arena. Then it costs one item to avoid getting stuck in the middle arena because there are no coins available there. Additionally, this makes obtaining an item a little more difficult because one needs to collect two items in order to gain one for the tower.

The costliest action is planting a **bomb** and is used to stop a tower from gaining a new level during the building process. It requires five coins per default and can only be placed adjacent to an opponent's tower. In the game settings one can decide if the bomb should also have the power to empty the storage of the tower. The explosion of the bomb has no effect on any of the characters.

The last action is reserved for **building** the next level of the tower. When all requirements are met (enough items stored and holding enough coins), the player can start a build process.

The process takes a random amount of time in an interval. The interval grows bigger the higher the level of the tower is. The upper bound increases in equal steps up to a maximal building time. The lower bound of the interval is fixed for the whole game. The default is 20 seconds, but this can be changed in the game settings to a value between 15 and 30 seconds. The maximal building time can be set to a value between 30 and 60 seconds (default 50 seconds). Only the last 15 seconds of the building process are indicated by a visible timer for the opponents to see (see also Subsection 2.1.3).

2.1.2 Flow of Actions

A game breaks down to the following steps. These steps are in no particular order (or by no means do they need to be repeated in this exact order).

- **Collecting coins**
  - Collect enough coins for the next moves to take
  - Consider that bombs are costly and many swaps may be required
- **Collecting required items for next level**
  - Swap to the middle arena, push correct button, wait for item(s) to spawn, collect them and finally swap out
  - Observe the movements of all characters and try to find opponent and kill him to get your required item by the amount of his current level
- **Store items in tower and start a building process**
  - Walk or swap to own tower
  - Store items
  - Start building process once enough items stored and enough coins in bag
- **Observe and intervene**
  - Observe other players and check their progress
  - Stop a building process of an enemy with a bomb

2.1.3 Status Display

Even though *Undercover Construction*'s purpose is to hide information some things are displayed on screen to assist each player's mind and help to find out more about enemies. The provided information during the game is illustrated in Figure 2.1.

On the left, information boxes for each active player are shown. The upper box shows the contents of the player's coin and item bags. The item bag shows symbols of all the items the player holds. The lower box shows the requirements for the next build level.

On the right, general information is shown that does not change during a play round. The costs for the action *swap* are shown in the upper box. The four symbols represent four swap cases with possibly different costs. A swap can start and end in the outer or inner arena, or it can have different start and end areas. The costs are configurable in the settings. The lower box shows the buttons and their respective actions as described in Subsection 2.1.1.

Adjacent to each tower there is a clock that starts counting down when the last 15 seconds of a build process are reached. In the example of the figure, one can see that Player 1 has started a build process and is 13.3 seconds from completing it.

Small symbols on the buttons and in the boxes connect the actions with the information shown. The wheelbarrow represents the items and coins the player carries around. The two dots with an arrow illustrate the swapping action. The crane is used to indicate the build button, the build requirements, as well as the build clock. The boxing glove and the bomb on the button icons refer to the actions attack and bomb, respectively.

2.2 Hidden Information

Undercover Construction gives the players the ability to hide themselves as well as their progress. The following two subsections give further details of which means the game provides the players to form strategies.

2.2.1 Location

At the start of the game, the players do not know which character they are controlling. They know, though, that they control a character close to their tower, which enables them to find themselves rather quickly.

During the game, players can hide among the **many NPCs**, which move either horizontally, vertically or diagonally. NPCs also stay still sometimes. If the player moves too smoothly an enemy can spot him, because his movements differ too much from the other characters on the field. NPCs only move to point of interests (POI)[1] at random. This can be useful for players to fool or distract others if they assume the NPC movements.

The **swapping mechanism** is good for fast traveling, but also needed to flee after a revealing action was performed, such as killing somebody or planting a bomb. Because swapping is done with characters in line of sight and several swaps might be needed to flee from a crowded corner, a player must make sure to have enough coins. When NPCs are swapped to the location of the player, they inherit the direction of the player for a short while, before they take over control. This makes it more difficult for an observer to notice when a swap occurs.

Item and coin pickups are delayed for two seconds. A vibration feedback helps the player to sense when a pickup was successful or not, because the visual feedback is not immediate. On the other hand, if there is no vibration, it is an indication that someone else took the item and must be close. NPCs do not pick up items and coins.

The players have **to expose themselves eventually**, because they either need to kill someone or swap to the middle arena to get their required items. **Attacking is visible** and one becomes prey for the others. Especially if the player guesses wrong and kills an NPC instead of another player: This is punished by losing all coins, and thus, the player cannot swap and flee as long as he does not collect new coins.

Inside the arena, a **visible button has to be pushed**. Because everybody knows the requirements of everybody, a long stay on a button is a clear indication that a character is controlled by a player. NPCs push buttons by chance as well, but eventually move away. Every two seconds an item appears in the middle, if any of the buttons is pressed.

A lucky player might push the button right before such a "spawn interrupt" and his item gets generated immediately. He has to wait two seconds on the button for each additional item. The spawned items in the arena have a **lifetime of ten seconds** before they disappear.

[1]Point of interests are for example the tower, the item buttons or the items and coins

This is to lower the chance that an NPC stays on a button for a long time and generating many of the items one of the player needs. The items are randomly placed somewhere on the green meadow. Waiting another two seconds for the chance of a closer item might be better than running across the meadow to reach the first item but bears the risk of attracting opponents.

2.2.2 Progress

Besides hiding the location, there are some ways to hide progress. In this game progress means collected items and the finished levels of the tower.

While inside the arena, the **item bag is not updated** visibly, because there are only a few characters inside and are therefore exposed and an additional visible aid is unnecessary. Once a player swaps out, the item bag is updated. Since this swap costs one item, the first item that the player collects is lost. This means that for a successful item pickup, the player is exposed rather long because he needs to collect two items. But naturally, the first item does not need to match his requirement and any spawned item can be collected right after swapping in. When a player thinks he has been spotted, fleeing to the outside is rather easy, even with this longer exposure.

The **items stored in the tower are not shown** and are stored permanently. The player needs to remember all stored items, or he runs the risk of collecting items for naught. Opponents can empty the storage of the tower with a bomb if the option is enabled in the settings. This gives a bigger incentive to use the bomb. On the other hand, if this option is disabled, players have a bigger incentive to collect and store items that are not required for the current level, because they can be used for another level.

A vibration feedback is given to the user when he successfully starts a **building process**. The process starts in the background where the required items are consumed and a random time is selected as described in 2.1.1. The last 15 seconds are counted down by the timer displayed next to the tower. Opponents can stop the process by planting a bomb. The bomb requires 5 seconds to explode, which means opponents need to be fast in order to stop the process. It is possible to stop the timer even from afar by swapping across the field, which gives the game a sense of action gameplay. But is better to **observe the course of action** in order to be ready when the timer starts counting down.

Because opponents react fast, they jeopardize to be found when the timer starts ticking. This gives the building player the opportunity to **defend his tower** by killing the approaching opponents. This is the only mean of defense, because a ticking bomb cannot be stopped.

It is worth to mention, that with such a kill, he gains items that are needed for the level that is currently building. This can mean two things:

a) build successful: the item is useless for the player, but he also did not receive a further advantage over the others with the kill

b) build stopped by bomb: the player will be able to build again sooner, because items for the next build were collected with the kill

Therefore, it is desirable for a player to stay close to his tower during a build process and kill approaching opponents.

2.3 Strategies

In the following, some possible strategies are explained. These strategies can be mixed and some might be better suited depending on the progress of the player. The pros and cons for each strategy are listed in Tables 2.1 to 2.4.

2.3.1 Sneak and Attack

Since all items can theoretically be obtained by killing opponents, a player can **avoid the vulnerable inner arena** altogether. But because attacking is very revealing, players will eventually get themselves into a vulnerable situation outside. Enough coins are needed to flee, after a successful attack. For an attack to work, the player must be close to the opponent and should be successful the first time, else he might get killed in reverse.

2.3.2 Run and Collect

This is a more active, **action oriented** way of playing the game. At first, many coins should be collected and the player swaps to the inside arena. Ideally, he collects any item that is already spawned inside. Then, he triggers a single item of his need without revealing his location for too long. Next, he collects the item, swaps out and stores it. This can be repeated until enough items are collected. This strategy is a good choice at the beginning of the game, because only one item is required for the first build. Later in the game it is wasteful, because each swap to the outside costs one item.

2.3.3 Wait and Observe

This strategy relies on **observing the non-hidden information**. This includes the displayed item and coin bags, the build timer, the attack animation, but possibly also the vibration of the controllers.

By observing the items of the opponents, a player can guess which player might start the next build process. Then he can wait for the player to approach his tower and kill him, or he can at least be ready to stop the building with a bomb. A player with many coins might indicate that he is using strategy *Run and Collect*. Since all players know the requirements of the other players, waiting close to the item button can lead to a successful kill.

When an attack is seen, one can try to find out several things. If a kill goes along with a drop to zero of someone's coin count, the player knows that this character is vulnerable and cannot immediately swap away. One can also identify the player exactly and know which level this player is at. So, one knows how many items one can gain by attacking this player and calculate if it is worth the risk and effort. If the kill is successful, one sees an increase of items in someone's item bag, also identifying the player.

Also, the noise of the vibration can be used to identify players. Listening to the vibration can help overcome the vanishing delay of picked up items. If a player suspects a character to be a player and follows him, he checks if the character interacts with the environment. Vibration feedback is immediate, whereas an item pickup might be visible too late and interactions with the tower give no visual aid at all. All of this works better when only two players play against each other, because keeping track of multiple vibrations is difficult.

2.3.4 Collaboration

Collaboration between players against someone else is a common strategy in games. In a local setting players need to **talk to each other** and the target can listen to the conversation. Nevertheless, it can be used in this game. When a player is close to finishing a game, players need to stop his build attempts with bombs. This is harder to prevent for the defender if he gets attacked by multiple players, even if he knows that he will get attacked. Inexperienced players can keep themselves alive when playing together. There is of course always the risk of getting betrayed by team players.

Pros	Cons
Can be used at any stage of play	Relies heavily on successful kills
	Fleeing is important

Table 2.1: Pros and cons of **Sneak and Attack**

Pros	Cons
Good starting strategy	Wasting items with swapping
Active and hard to follow for others	Ignoring opponents

Table 2.2: Pros and cons of **Run and Collect**

Pros	Cons
Adaptive	Passive and relies on others
Useful in any situation	Hard to keep track

Table 2.3: Pros and cons of **Wait and Observe**

Pros	Cons
Stop good players late in the game	Risk of betrayal
Chance for inexperienced players	Opponents can hear strategy

Table 2.4: Pros and cons of **Collaboration**

2.3.5 Advanced tricks

The following tricks can be used with any of the strategies:

- Do not start building immediately and collect not currently required items. Allows to build levels consecutively which is surprising.
- Collect more items if the course of events allows it and do not store all of them. Since the oldest items is used for swapping out of the arena, one is more agile in the middle arena.
- Try swapping with a suspicious character. If the swap does not work, the character is controlled by a player.
- Place yourself on a button and swap with NPC inside the arena. Needs NPC to be in line-of-sight and its movements on the button afterwards are unpredictable.

Chapter 3

Implementation

This chapter presents some interesting parts concerning the implementation of the game.

3.1 Unity3D

The game is developed using Unity3D[1]. Unity is a cross-platform[2] game engine. A free personal license can be obtained and is therefore a great way to start exploring game design. Numerous tutorials, good documentation and a large community facilitate game development for beginners.

C# is the high-level language used for programming scripts. The editor provides pre-designed objects and components, which are highly customizable. Objects are placed into scenes which can represent for example a single level or a settings menu.

Undercover Construction uses only two scenes: the title screen and the game scene. The title screen is used for the joining routine for players who want to participate. A settings menu is accessible as well and is loaded inside the same scene. In the settings, some variables can be changed, such as the number of NPC characters or the maximal required time for a tower to build its next level.

When the player starts the game, the scene for the game is loaded. The information from the title screen can be shared and is used for the preparation of the game scene.

3.1.1 Raycasting

Raycasting, and in *Undercover Construction*'s case *Physics 2D Raycasting*, is a powerful tool to implement interaction between objects. Unity describes the process as follows:

> "**A raycast is conceptually like a laser beam that is fired from a point in space along a particular direction. Any object making contact with the beam can be detected and reported.**" [4]

Raycasting was used in this project for three things (see Figure 3.1):

1. Interaction with tower for storing and building
2. Collision avoidance for NPCs
3. Swapping characters

[1] www.unity3d.com

[2] Unity supports 27 platforms, such as Windows, Mac OSX and Linux

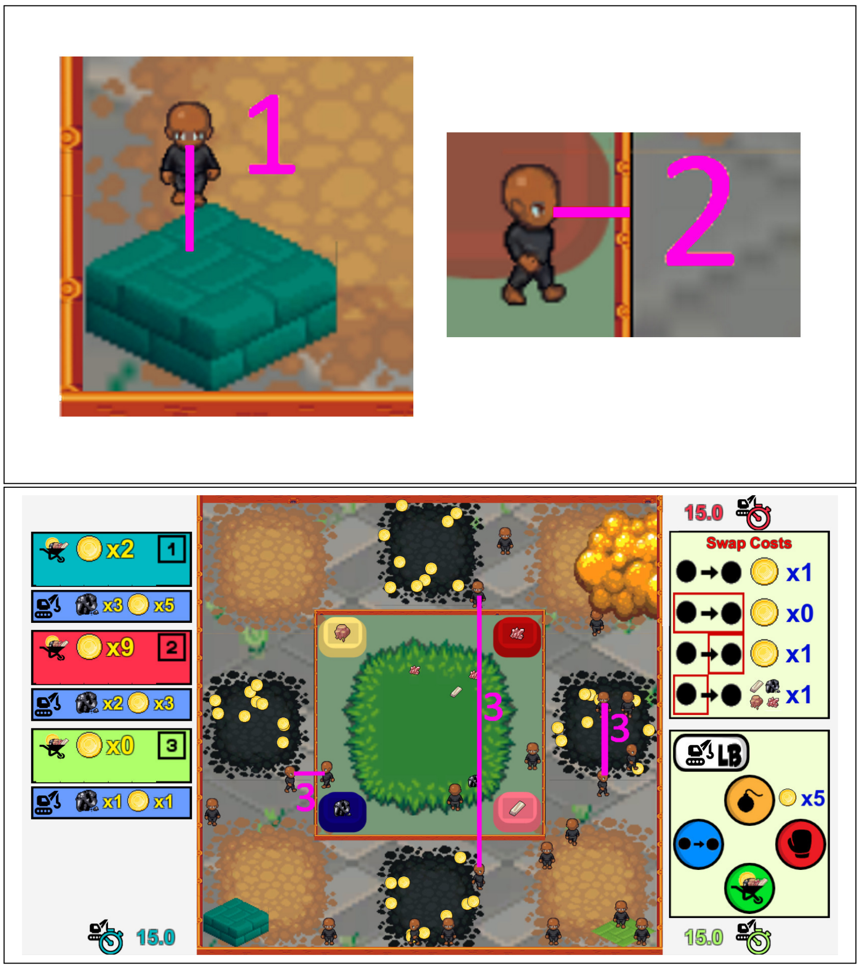


Figure 3.1: Top: Schematic representation of applying raycasting 1) to interact with objects and 2) to detect walls for collision avoidance. Bottom: Use of raycast for swapping (from left to right): Swapping to inner/outer arena, swapping across field and simple swapping across short distances

Rays are cast in the direction the character is currently looking. The programmer receives an array holding all the physical elements the ray hits along the way. Layers can be defined and put on each object in the environment. A ray can then be instructed to only interact with objects on the defined layers.

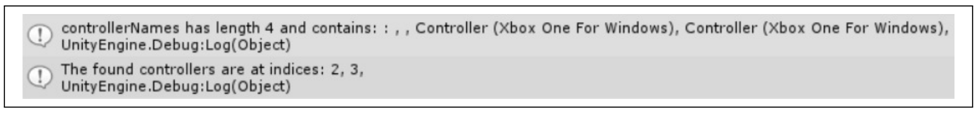
For **interaction** with the tower, a layer *tower* is defined and set on the tower objects. When the player asks to store items in the tower, a ray is cast and travels a certain distance the programmer chooses. The distance can therefore be used to define how close a player needs to be in order to perform the store action. If the player is close enough, he will receive the tower object on which he can request to store the items he is holding.

Collision avoidance is similar in concept. NPCs continuously cast rays and will notice when a wall is in front of them. The returned object also contains information about the distance and the location of the other object. NPCs can then be instructed to change direction if some conditions are met. **Swapping** can easily be implemented by replacing the position of the raycasting player with the position of the NPC that got hit, and vice versa.

3.2 Controller Assignment and Vibration

During development, different problems concerning the controls and vibration occurred. For the controls, Unity's out-of-the-box support for gamepads is used with the *Input Manager*[3]. The vibration feature is enabled with a C# wrapper around XInput for .NET applications (such as Unity3D), because Unity does not provide easy to use vibration capabilities with controllers. The wrapper is called *XInputDotNet*[4] and is open-source under the MIT License.

Figure 3.2 shows that the detected controllers are not always recognized as Controller 1, Controller 2, etc. In the example of Figure 3.2, Unity would refer to the two controllers as *Joystick 3* and *Joystick 4* for the controllers at array indices 2 and 3, respectively. Detecting the input of *Joystick 1* for player one would always return an empty value. To solve this, the recognized controllers are detected and a map from player number to joystick number is generated.



```
controllerNames has length 4 and contains: , , , Controller (Xbox One For Windows), Controller (Xbox One For Windows),
UnityEngine.Debug:Log(Object)
The found controllers are at indices: 2, 3,
UnityEngine.Debug:Log(Object)
```

Figure 3.2: Debug output showing that the attached controllers might not start at index zero

XInputDotNet's vibration method uses the so called *PlayerIndex* to associate the vibration command to the controller. As it happens, the index of a controller in Unity (as mentioned above) and the PlayerIndex might not match. This mismatch is solved by a join routine in the title screen as depicted in Figure 3.3.

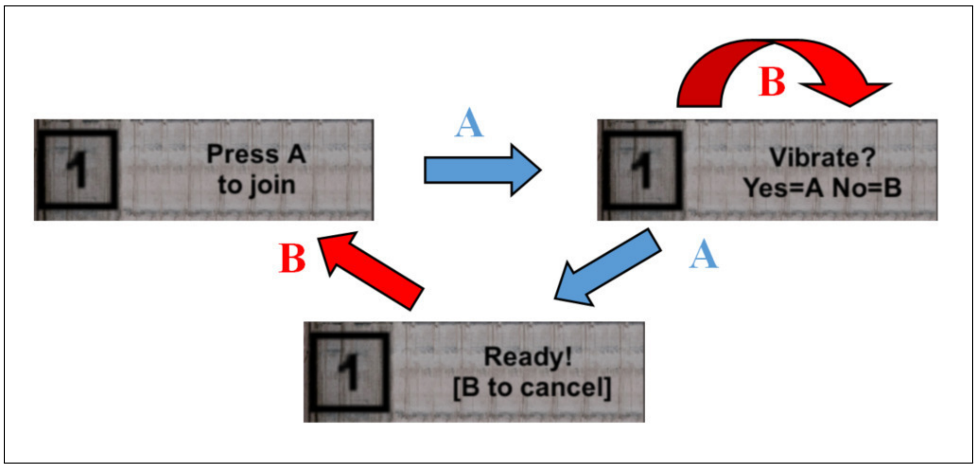


Figure 3.3: Join Routine: A player has to press **A** to join. He is asked to press **B** until his controller vibrates. After confirming with **A**, the player is ready. With **B** he can undo the join.

To avoid the above problems it might be desirable to ignore Unity's input management and build a custom solution exclusively using *XInputDotNet*. How this can be done is described in an online tutorial [5].

[3]https://docs.unity3d.com/Manual/class-InputManager.html
[4]https://github.com/speps/XInputDotNet

Chapter 4
Discussion

Undercover Construction uses different concepts to integrate asymmetric information in the gameplay. The straightforward approach is to not show any information on screen. As soon as multiple gameplay elements come together, it is not practical to conceal all information, because players get overwhelmed.

This works shows that some trade-offs must be made and that there is room between the two extremes "show all" and "show nothing". Delays in displaying information introduces an asymmetry in information, because an observer cannot easily link cause and effect. The responsible character can already be far away after an item disappears for example.

Play rounds with friends showed that the game has to be explained thoroughly for a first play. The mix of hidden and disclosed information is unusual and people are not used to it. It is best to play a test round first and explain some possible strategies, because experienced players are in a significant advantage. The good news is that players get accustomed to the gameplay fast.

The following section explores how the ideas presented can be expanded.

4.1 Future Work

A video game is never really finished, and extensions are always possible. Many ideas to adapt and extend *Undercover Construction* have been collected during development.

Representations of the **world wonders** could replace the simple towers. Each wonder could be requiring different resources and could be equipped with unique powers (ideally using hidden information). Taking the "Great Pyramid of Giza"[1] as an example, one level to build might be the grave chamber which needs a dead body as resource and not a conventional item like stones. As a special power an Egyptian god might be called to destroy a level of an opponent's wonder.

The **combat system** in *Undercover Construction* is as simple as it gets and leaves room for improvement. Straightforward modifications could be a health system with a possibility to defend in contrast to the one-hit-kill principle of *Undercover Construction*. Variations of attacks could be implemented, such as attacks from afar by using thunderbolts or the like. These ideas could be linked well with the powers of the wonders as suggested above.

As a mean of defense the idea to **create walls** is promising, because it gives a range of applications during gameplay. First, it could be used as a shield against an approaching character and second, it could defend the tower against bombs. Furthermore, a player could build walls all around him and shield himself from any attack for a while. This is useful

[1] https://en.wikipedia.org/wiki/Great_Pyramid_of_Giza

when standing on an item button to spawn multiple items or simply when one desires to collect coins in peace for a while.

In support of the wall idea above, a **clock item** comes to mind. The wall must expire sometime. To overcome this, a clock item could be introduced that can be used on a wall to extend its lifetime. The clock might also come in handy to decrease the build time of the player's own tower or increase the build time of the opponent's tower. As a new hidden information mechanism, the developer can allow the use of the clock on an enemy tower at all times, such that the player cannot know, how long the next build will last.

An **adaptive playfield** could enforce other mechanics in the game. For example, could the field gradually get destroyed with cracks in the floor which one can only overcome by swapping. Clouds could pass by and give players an opportunity to hide under it.

The vibration issues described in Chapter 3.2 sometimes caused that the rumbling of two controllers got mixed up. This could be used as a game mechanism where each **player feels the vibrations of another player** and he is only allowed to kill said player. So, observing becomes twofold: First, find a character that is not an NPC and second, try to figure out if its actions match the vibrations you feel.

An ambitious concept for combat is the integration of **mini games**. The reward for winning a mini game is an item. Imagining the middle arena as a second screen running games within the game, a player has to enter the screen and win the short game that will start there. If no other player enters the screen within a countdown, either no game will start and the player gets the item for free, or he will need to play a single player game such as a puzzle. On the other hand, if more players enter the screen, a multi-player game for two, three or four players will start. The mini games could be as simple as the well-known Rock-Paper-Scissors game or more sophisticated like a top-view car racer like *Retro Racers*[2]. The conceptional photo collage in Figure 4.1 shows how this could look like with Rock-Paper-Scissors. Certainly, many mini games of existing games could be adopted. Nintendo's mini game collection series *Mario Party*[3] and *Wario Ware Inc.*[4] come to mind.

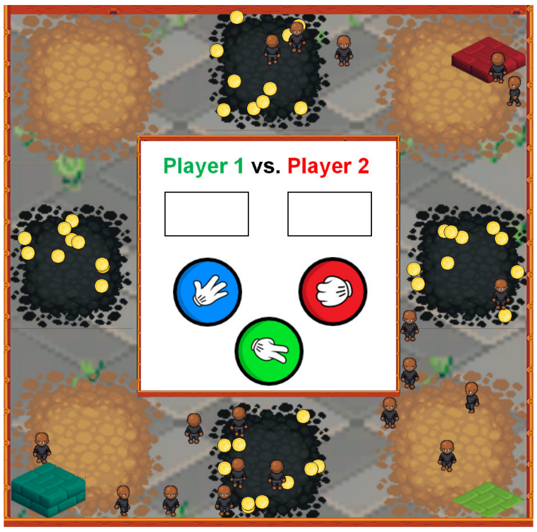


Figure 4.1: Concept showing that the middle arena could be used for mini games such as Rock-Paper-Scissors. Here: Player 1 and Player 2 fight over the win of an item.

[2]https://loadupgames.itch.io/retro-racers
[3]https://www.mariowiki.com/Mario_Party_(series)
[4]https://www.mariowiki.com/WarioWare_(series)

Bibliography

[1] Josh Glazer, S.M.: Overview of Networked Games. In: Multiplayer Game Programming – Architecting Networked Games. (2015)

[2] Wikipedia: Gameplay. In: https://en.wikipedia.org/wiki/Gameplay. (2018)

[3] Davison, A.: An Isometric Tile Game. In: Killer Game Programming in Java – Java Gaming & Graphics Programming. (2005)

[4] Unity: Unity Documentation – physics2d.raycast. In: https://docs.unity3d.com/ScriptReference/Physics2D.Raycast.html. (2018)

[5] McCauley, L.: Xbox360 Gamepad Input & Management – Unity Tutorial. In: https://lcmccauley.wordpress.com/2015/04/20/x360-input-tutorial-unity-p1/. (2015)