PLAY AREA UTILIZATION OPTIMIZATION FOR ROOM-SCALE EXPLORATION OF VIRTUAL WORLDS

A Thesis

presented to

the Faculty of California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Electrical Engineering

by

Chase VanderZwan

June 2023

COMMITTEE MEMBERSHIP

TITLE:	Play Area Utilization Optimization for Room-scale Exploration of Virtual Worlds
AUTHOR:	Chase VanderZwan
DATE SUBMITTED:	June 2023
COMMITTEE CHAIR:	Christian Eckhardt, Ph.D. Professor Computer Science
COMMITTEE MEMBER:	Michael Haungs, Ph.D. Professor Computer Science and Software Engineering
COMMITTEE MEMBER:	April Grow, Ph.D. Associate Professor Computer Science

Abstract

Play Area Utilization Optimization for Room-scale Exploration of Virtual Worlds

Chase VanderZwan

Virtual Reality (VR) opens up new possibilities for developers to create immersive worlds and experiences. While it's possible to craft unique and engaging interactive environments with unprecedented realism, the virtual world is constrained by the real one. Current approaches to player navigation in VR applications include joystick controls, teleportation, and motion-based movement. While these methods are effective in certain scenarios to overcome real-world limitations, my research introduces a novel approach that leverages room scale-based movement, with portals, to traverse a given VR world. This work presents algorithms that accurately predict the percentage of play area utilized, and rules to implement typical game elements to allow large scale virtual immersion under real world constraints.

Contents

List of Figures

Chapter 1

Introduction

In recent years, virtual reality (VR) has emerged as a platform with immense potential for delievering immersive experiences. However, in order to fully leverage VR's capabilities, it is essential to understand how to create user experiences that align with their expectations and desires. A significant challenge to achieving VR immersion is minimizing visually induced motion sickness (VIMS) while maximizing user immersion, or sense of "presence". The choice of locomotion technique plays a crucial role in addressing these challenges.

Researchers Bolestis and Chasanidou (2021) have highlighted the increasing research focus on VR locomotion techniques, emphasizing their importance in the field. They have identified four key locomotive types: Motion-Based, Room scale-based, Controller-based, and Teleportation-based. Among these, walking-in-place (WIP) and teleportation-based movement have emerged as the two most popular movement techniques in modern VR applications [3].

The Point & Teleport technique widely adopted due to its ease of use and familiarity. While this technique offers reliability, it can sometimes detract from the immersion factor. Motion sickness can occur since the user remains physically still while the virtual world moves around them [4].

Many existing VR applications predominantly rely on joystick or teleportation movement methods. It provides a quick and precise means of transportation through a virtual world, reminiscent of traditional controls in non-VR video games. Consequently, this method often compromises certain aspects of user immersion. Studies have shown that controller movement, namely joystick controls, can induce motion sickness in users in many cases [9]. Although similar to Point & Teleport in the fact that both methods involve the virtual world moving while the user remains stationary, joystick controls often are reported to induce more nausea due to the continuous motion [5].

Further research conducted on locomotive taxonomies, such as controller/joystick, gesture-based, and teleportation, suggests that artificial interaction can lead to a less physically intense experience, potentially increasing cognitive load and, consequently, motion sickness due to a disassociation between the user and their environment [1] [10] [2].

The Walking-in-place movement technique, similar to room scale-based locomotion, has been demonstrated good immersion potential. While some users find the physical aspect to be tedious or tiresome, others find this feature to add some entertainment or fun to the VR experience [2].

Furthermore, there are various other techniques that offer unique solutions and options to the locomotive conversation. One such technique is Redirected Walking (RDW), which attempts to rotate the user towards the center of their real-world play area using eye saccades. This techniques can help facilitate a feeling of infinite space to roam about, despite the play area constraints [6].

One of the limitations of Redirected Walking is the required play area size. One study found that, in order for RDW to go undetected, the user must be

redirected in a circular arc with a radius of at least 22m [11]. However, more recent studies show that this requirement can be reduced if the player is following a curved path, rather than straight, in their virtual environment [8].

Transition techniques, alongside locomotion, are vital components to the VR user experience and contribute to the overall sense of immersion. One study [7] investigated six transition techniques which included:

- Fade: Vision fades to black then the new scene fades in.
- Dissolve: Two scenes dissolve continuously into each other. A common transition used in film.
- Transformation: A spatial transition similar to Dissolve, but occurs all around the user.
- Portal: A flat plane which can be peered through. Walking through the portal transports the user to a new place in the virtual world, seamlessly.
- Orb: A spherical portal that can be grabbed by the user. When brought close to the user, is wraps around the user's head, seamlessly transitioning to another area of the virtual world.

This study found that users reported a higher sense of presence when using the Orb and Portal techniques due to their high degree of interactivity. The Slater-Usoh-Steed Presence Questionnaire [12] was employed to measure presence. Additionally, these two techniques scored high in ratings of continuity. Users responded positively in the ability to preview the next scene and anticipate their next position before the transition completed.

In this work, I introduce two algorithms designed to enhance the utilization of the user's play area for seamless exploration of virtual worlds. By leveraging

room-scale movement and portal mechanics, players can enjoy uninterrupted, immersive experiences within the virtual environment.

The first algorithm focuses on providing developers with valuable insights into how the placement of portals in the virtual world impacts the available play area. This information enables developers to optimize level design according to their specific objectives. For instance, in certain scenarios, developers may choose to restrict access to certain parts of the play area to create challenging gameplay experiences. Conversely, in other situations, developers may prioritize maximizing the accessible area to promote exploration and freedom.

The second algorithm is a powerful tool that complements the first algorithm by identifying portal locations that integrate with the virtual world's structure and layout. The algorithm assists developers in determining suitable portal locations that fulfill customizable requirements. Employing this algorithm allows developers to create compatible custom game elements in their game layout with this game type.

Chapter 2

Background

2.1 Play Area

Most VR applications require the definition of a boundary, or *Guardian*, which designates where the user can walk. Features such as the Oculus Guardian System or Valve's Chaperone allow users to mark off walls and oftentimes objects, such as tables or couches. In some cases, these marked off objects then appear in the virtual world, but largely this method is used as a means of indicating to the user when they are approaching the edge of their boundary. Oculus requires a minimum of a 3ft x 3ft area while suggesting 7ft x 5ft as the optimal size. This area can be marked while in the headset, using controllers to draw an area, or synthesized with the help of external equipment, such as Valve's Base Stations. These scan your room with sensors to draw a play area. Moving forward, any reference to this marked off area will be with the term **Play Area**.

Additionally, users can choose between a stationary or room-scale play area. Play area's built in stationary mode assume an approximate circle boundary with the assumption that the user will remain in-place while using their headset. Room-scale mode refers to methods explained previously. That is, explicitly marking off an area of space. This method allows users to freely move about

their play area in the real world, which translates to movement within the virtual world as well. The room-scale setup will be required for all algorithms and implementations in this work.

2.2 Portals

A core game element used throughout this work is portals. Throughout the gaming industry, many variants of portals have been used. Portals have the potential to grant players increased freedom in how they interact with the given virtual world. There are many implementations for how portals can be placed in the world and interacted with. In this work, the use of portals increases the range at which room-scale movement allows players to move. Consequently, when a player uses a portal, the offset between the virtual and real world may change. For these reasons, in order to more clearly demonstrate what my algorithms provide, portals have been restricted to the following implementation.

First, they can only be entered from one side. Each portal is a plane, with one surface. Second, portals will be shot from a portal gun on the player. If shot onto a valid surface, the portal will snap into place on that surface. All portal-able surfaces exist in cardinal directions.

2.3 Software

2.3.1 C++

All algorithms presented in this work were made in C++ using Visual Studio 2022. The library *matplotlib* was used for visualizing results. Additionally, the library *GLM* was used for various data structures such as Vector2.

2.3.2 Unity Game Engine

Integrated into the workflow of this work was the use of Unity Game Engine to create a technology demonstration. The project uses Unity Editor version 2021.3.12f1 and makes use of Unity's Universal Render Pipeline. The demo makes use of Tom Goethal's *Portals for VR* asset which can be found on the Unity Asset Store. This package provides the logic and game elements to create portals that work with a VR headset. During use, the asset was at version 1.0.8c. Under this version, the Unity project must be run with single-render pass instancing under Unity's *XR Plug-in Management* in Project Settings. The OpenXR plug-in was used, as opposed to Oculus. All scripts were made in C#, as Unity's language of choice.

2.4 Hardware

2.4.1 VR Headset

Throughout this work, all testing was performed with an Oculus Quest 2 VR headset.

Chapter 3

Methodology

3.1 Play Area Representation

A user's play area, or guardian, can be defined at a variable size and any shape. For the sake of consistency, any reference to a play area in this work is assumed to be approximately a square shape, or one that can fit a square shape within it. All tests in this work were performed with an assumed play area of approximately size 8' x 8', but in theory all algorithms should work at any square size, with minor adjustments. With this assumption, my system is directly compatible with a 4x4 Tile-based environment representation. To navigate the world, users must walk around their room-scale guardian. The use of portals allows for movement between Grids. No joystick or point teleportation is used.

Figure 3.1: Play Area with Labeled Tiles

Figure 3.1 demonstrates the notation my system follows. A 4x4 platform divides the user's play area into 16 Tiles labeled as shown. Moving forward, any 4x4 platform will be referred to as a *Grid*. The North-facing arrow on Tile 1 is a reference point for the platform's true direction. My algorithms have several processes which require symbolically manipulating Grids to obtain information. This Grid structure is critical to the structure of my algorithms.

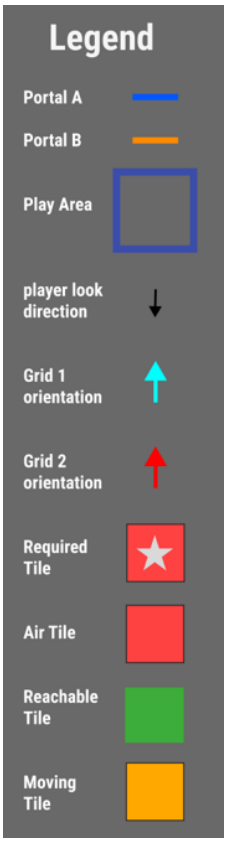


Figure 3.2: Legend for Grid Notation

When describing images in the format of Figure 3.1, refer to Figure 3.2 as a guide. Two portal and orientation colors are provided in order to differentiate images. Unless otherwise specified, the blue colored orientation and portal are used to describe source elements, whereas orange describes destination elements. The Play Area box represents what the user's play area encompasses in the virtual world. Finally, the player look direction arrow indicates the direction the player will be looking when using a portal.

3.2 Data Structures

3.2.1 Grids and Tiles

In my representation, the world is built up of many 4x4 Grids. The use of a 4x4 dimension allows for the demonstration of many interesting complex interactions made possible with my algorithms. Smaller dimensions limit this complexity, whereas larger spaces may allow for more complex game elements. In order to fully represent each 4x4 Grid object, my system defines a Grid as a structure containing a 2D array of *Tiles* along with a reference to a Play Area Offset, in a Vector2. Tiles are custom structures defined as int arrays of size 4, representing *North*, *East*, *South*, *West*, in that explicit order. Each index of a Tile is represented by a macro. These marcos define the qualitative data that exists on that Tile's surface. A visualization of the Grid and Tile structure made in C++ can be seen in Figure 3.3.

Figure 3.3: C++ Debug Visualization of a 4x4 Grid of Tile structures using Macro flags

3.2.2 Portal-able Walls

Along with Grids and Tiles, the other important virtual game object are walls in which the player can place a portal. Similar to Grid dimension, the implementation of portal-able wall placement can largely be customized by developers. In my implementation, portal-able wall placement is determined by Tile. Once a Tile is marked as a **Pillar Tile**, portal-able walls are placed on all four edges of that Tile.

3.2.3 Macros

Any given Tile within the system contains information about each of its four edges. To ensure efficient and readable code, I have utilized several macros as integer flags. The macros employed in this project include the following:

- **PERIMETER**: Represents Tile edges located on the perimeter edges of a given Grid. When defining a Pillar Tile, these values help determine which walls should be marked as *Iron* (invalid).
- **PORTALABLE**: This macro identifies edges where virtual portal-able walls have been placed.
- **IRON**: Denotes edges that have a virtual iron wall. Iron walls do not allow portals to be shot on them. These surfaces are often used to demonstrate invalid portal locations or restrict portal placements in specific scenarios.
- **PORTAL**: Reserved for the portal surfaces themselves. It is implied that a virtual portal-able wall exists here as well. The direction of the portal surface can also be derived via the index of this macro. For example, if a

portal macro exists at index 1 of a Tile, the portal surface faces East. It is important to note that only two portal macros can be present in the virtual world with a maximum of one per Grid.

- **DEFAULT**: Indicates an empty Tile edge (No walls or special meaning).
- **MOVING**: Any Tile which moves must have all edges set to this macro.
- **AIR**: Represents the absence of a Tile at that location. Overrides flags such as perimeter. Useful for creating custom dimension game elements while maintaining a compatible dimension.

By leveraging these macros, calculations are optimized and code readability is improved. For instance, by defining perimeter surfaces during Grid instantiation, default invalid portal locations can be easily identified, and the appropriate placements of iron walls can be immediately known. Some flags, such as Air or Moving, which pertain to the entire Tile, could be replaced if an additional identifier is added to the *Tile* structure. Figure 3.3 showcases the use of these flags. In this case, 5's refer to perimeter edges, 1 represents portalable walls, and 8 denotes the portal surface location. Additionally, two Pillar Tiles, Tile 6 and 11, are present with a portal on the south surface of Tile 6. For a visual representation, refer to Figure 3.4 and 3.5 which depict the Grid in the accompanying tech demo.

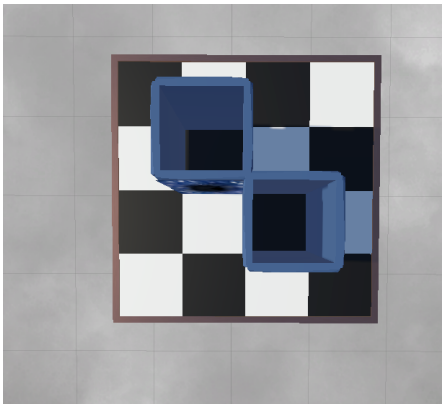


Figure 3.4: Top Down Visualization of Fig 3.3 in Unity

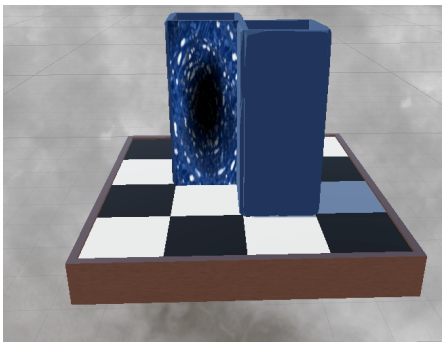


Figure 3.5: Front Visualization of Fig 3.3 in Unity

Finally, Figure 3.6 shows this example in Grid notation. Note that my Grid notation only worries about displaying portal location and, when applicable, the Play Area Offset.

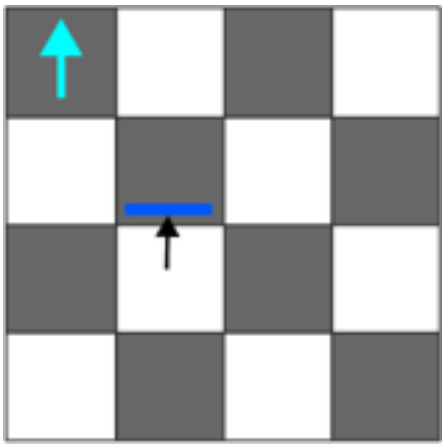


Figure 3.6: Grid Notation of Fig 3.3

3.2.4 Custom Game Elements

In my research, I have developed 4x4 Grid structures that provide users with the ability to traverse any distance within the virtual world. However, it is crucial to demonstrate how these structures might interact with game elements designed for conveyance purposes. To address this, I used a new structure called the *Moving Element*, which follows a similar framework as the Grid but consists of a 3x3 array of Tiles

I have defined my *Moving Element* structure to be comprised of one Pillar Tile, one Moving Tile, and seven Air Tiles. Additionally, I have marked three nearby tiles as *Required*, which the next section will elaborate on. This structure will serve as an example of how custom game elements can be compatible with 4x4 Grids (or whichever dimensions a developer has defined to be their foundation). One example layout of this structure can be seen in Figure 3.7.

In practice, custom game elements can be formatted in one of two ways. First, they can directly align with the dimension of the Grid, with Air Tiles used in appropriate places to create a custom dimension. Alternatively, they

can be explicitly defined with their specific dimensions. However, it is important to note that when using the latter approach, special attention must be given to convert indices between the two structure dimensions. This consideration becomes particularly relevant when discussing the *Compatible Portal Location* Algorithm, which will be addressed later in this thesis.

3.2.5 Required Tiles

When working with custom game elements, the key to compatibility is marking *Required Tiles*. Required Tiles are those which must be reachable by the user after they enter said custom game element. In my *Moving Element* example, there is one moving Tile that moves in a known direction when activated. Naturally, it might be desired that the player have some guaranteed reachable Tiles once they dock at the next Grid or game element.

When discussing Grid Alignment, the concept of required Tiles will become more clear. In brief terms, "reachable" refers to Tiles that exist within the user's play area. When a player portals to another Grid or game element, the play area will encompass a different set of Tiles. If certain Tiles are marked as required, this sets some requirements for where the user should portal before reaching the associated game element. The use of required Tiles will be more relevant when discussing the *Compatible Portal Location* Algorithm.

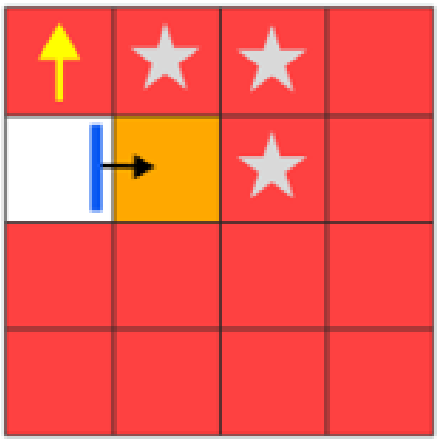


Figure 3.7: Grid Notation for a Custom Element Structure

In Figure 3.7, the Grid notation is presented to represent an example custom game element. The red Tiles correspond to Air Tiles, indicating empty spaces. The orange color refers to a Moving Tile. White or gray Tiles represent Pillar Tiles, where the player can portal onto. An orientation arrow is included to mark the absolute rotation of the game element, similar to the 4x4 Grid. Finally, stars are added to some Air Tiles to mark which are *required* to be walkable.

In total, the figure showcases a custom element with a 1x2 dimension consisting of one Pillar Tile and one moving Tile. After the moving Tile docks into the next designated game element, the three marked required Tiles must be reachable within the player's play area.

3.3 NTA Algorithm

3.3.1 Overview

In the context of the game type I have presented in this paper, at any given moment, the player will be standing on some game element (Grid or otherwise).

The player's play area, assumed to be 8ft x 8ft, encompasses some number of Tiles within this current game element, which I will refer to as a **Source** game element. From here, the player may move to another game element, which I will refer to as a **Destination** game element. The player's play area might then encompass a new number of Tiles. I have labeled this quantity of Tiles as **NTA**, or Number of Tiles Available. The term *Available* implies that only the Tiles that exist within the bounds of the play area will be reachable by the player.

To calculate the NTA of a given game element, namely future game elements, my NTA Algorithm can be used. The NTA metric is an insightful metric that allows developers to evaluate available space at select portions of the map and explore potential gameplay scenarios.

The NTA metric becomes particularly valuable when evaluating potential portal sequences. In the accompanying tech demo, each Grid randomly generates some number of Pillar Tiles at runtime. Here, the NTA Algorithm can be used to test each combination of possible portal locations, using every combination of Pillar Tiles across consecutive game elements. This information holds significance for a range of use cases, some of which include:

- Level Design Optimization: Analyzing the NTA of different game elements can help developers understand how players might utilize the available space and optimize level design for enhanced gameplay experiences.
- Player Movement and Navigation: Calculating the NTA for various paths and routes assists in designing player movement and navigation, ensuring sufficient space for strategic maneuvering.
- Challenge and Difficulty Balancing: NTA can be used to balance the difficulty of levels by identifying areas with limited or expansive space, enabling

developers to adjust level design accordingly.

- Puzzle Solving and Exploration: NTA aids in designing puzzles and exploration-focused gameplay by ensuring sufficient space for players to interact with puzzle elements and explore the environment effectively.

By leveraging the NTA metric and the capabilities of the NTA Algorithm, developers can make informed design decisions, optimize gameplay experiences, and create engaging levels that maximize the utilization of available space.

At a high level, the NTA Algorithm is performed in six steps.

- Step 1: Define reference to two game elements. Locate necessary information such as portal locations and the Play Area Offset on the source game element.
- Step 2: Align the source to the destination game element using the portal locations.
- Step 3: Calculate the game element offset.
- Step 4: Using the game element offset, calculate the Play Area Offset for the destination game element.
- Step 5: On the destination game element, determine which Tiles exist outside of the newly defined play area.
- Step 6: Derive NTA.

3.3.2 Game Element Alignment and Offset Calculations

The first step towards calculating NTA is aligning two game elements. To do this, the following information is required:

- **Portal A**: Portal on game element 1 (current/Source Grid). Tile location and surface (N, W, E, S)
- **Portal B**: Portal on game element 2 (Destination Grid). Tile location and surface (N, W, E, S)

As a baseline, let's start with two Grid game elements. The alignment and offset calculation process requires the following steps:

- Step 1: Determine source and destination portal surface direction.
- Step 2: Calculate the number of 90-degree clockwise rotations of the Source Grid needed to make it such that the two portals are facing opposite directions.
- Step 3: Find the Forward Tile of the Destination Grid and the *Portal Tile* of the rotated Source Grid.
- Step 4: Using the these two locations, calculate the offset between them in terms of rows and columns. Use the absolute orientation of the Destination Grid as reference.

Aligning the Grids in this way effectively removes the gap between the two portals and allows us to analyze the direct spatial relationship. Essentially, instead of teleporting the player to the next portal location, we are shifting the Source Grid such that it places the player at the destination directly. This is imperative as it allows us to see how the user's play area impacts the destination game element.

Portal Surface Direction

In my implementation, the NTA algorithm is assumed to run as an analysis tool in the absence of live gameplay. As such, portals are placed symbolically. In the case of two Grid game elements, the algorithm loops through all combinations of portalable wall surfaces and places a portal on each.

In the C++ implementation, the portal surface location can be directly derived via its index in the Tile structure. For example, if the portal macro exists on Tile 6, index 3, as is the case in Figure 3.3, then the portal is facing South. Listing 3.1 demonstrates how the Tile index of each portal can be used to calculate number of clockwise rotations. Note: In Unity, this can be simplified by grabbing the forward vector of each portal game object.

Listing 3.1: Calculating Rotations

```
int Rot_A = 2; // Portal A Surface = Tile[2]
int Rot_B = 3; // Portal B Surface = Tile[3]

int Rot_B_Flipped = (Rot_B + 2) % 4;
int rotations = (Rot_B_Flipped - Rot_A) % 4;

// Transforms counter-clockwise to clockwise
if (rotations < 0)
    rotations += 4;

return rotations;
```

Rotating the Source Grid

Once the number of rotations is found, we can perform the first step of alignment. That is, rotating the Source Grid about its center. Listing 3.2 demonstrates how this update is performed for a 2D array. Since each element of the 2D array is a Tile structure, it is necessary to rotate these elements individually as well. In my implementation, rotating a integer array of size 4 is as simple as the following: `newIndex = (index + 1) % ArraySize`. Following this, macros must be updated. For example, perimeter flags must be removed and re-computed.

Listing 3.2: Grid Rotation

```
Tile tempGrid[4][4];
for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        tempGrid[j][3 - i] = _grid[i][j];
    }
}

for (int i = 0; i < 4; ++i) {
    for (int j = 0; j < 4; ++j) {
        _grid[i][j] = tempGrid[i][j];
    }
}

for (int idx = 1; idx <= 16; ++idx) {
    rotateTile90Clockwise(idx);
}
```

Game Element Offset Calculation

After updating the orientation of the Source Grid, the next step is to determine the translational offset. This is calculated using the Forward Tile of the Source Grid and the Portal Tile of the Destination Grid, or vice versa.

The **Forward Tile** refers to the Tile in front of the portal surface. In other words, the Tile the player would be standing on after portaling. There is one Forward Tile for each portal. The **Portal Tile** is simply the Tile containing the portal flag.

Aligning the rotated Source Grid's Forward Tile and the Destination Grid's Portal Tile effectively places the portals against each other's back. One important distinction is that the Source Forward Tile location must be in terms of the Grid's new orientation. For example, let's say the Forward Tile for a given Source Grid is Tile 2. If alignment requires one 90-degree rotation, the Tile we want for the Grid Offset calculation is Tile 8.

Using the (row, column) location of each target Tile, we get a Vector2 Offset value which describes the relationship between the two game elements.

Play Area Offset Calculation

While the *Grid Translation Offset* is an important metric, it cannot be directly used to determine what Tiles are reachable. This is because the play area is our only consideration for determining reachable Tiles. Abstractly, the algorithm must be able to calculate the NTA of a Destination Grid given any Source Grid layout accompanied by any Play Area Offset.

The Grid Offset calculation's purpose is to update the **Play Area Offset**,

or PAO. After the player portals, the user's play area now pertains to the new Grid, or other game element. The Grid Offset allows us to find where the play area now exits on this new Grid.

The user's Play Area Offset to the Source Grid abides by the same transformations as the Source Grid in the previous step. For each rotation, the Play Area Offset must also be rotated once. After the rotations are considered, the Grid Offset is added.

The offsets can be calculated in either direction, from the Source Grid to the Destination Grid or vice versa, as long as the calculation is consistent. In my implementation, a Play Area Offset of (-1, -2) would indicate an offset such that the bottom row and two rightmost columns of Tiles are unreachable by the player (Figure 3.8). w

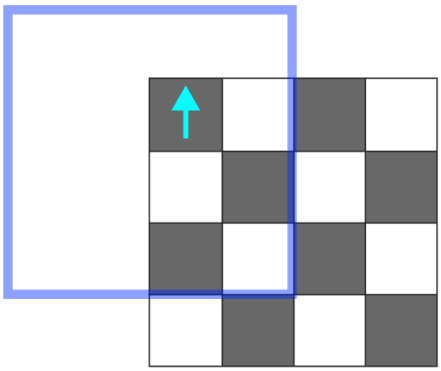


Figure 3.8: Demonstrates a Play Area Offset of (-1, -2) to a Grid, rendering the all Tiles save the bottom row and two rightmost columns from being reachable by the player

3.3.3 NTA Calculation

Once the Grid Offset and Play Area Offset are known, the NTA calculation is simple. In my implementation, I used a bitmap called the **Reachable Tiles Bitmap** that has a size equal to the number of Tiles in a given Grid, in this case

16, as seen in Figure 3.16. The final consideration is the destination Portal Tile. In my case, the Pillar Tile design means there will always be at least one extra Tile that is not walkable. Abstractly, consider any virtual objects obstructing your player from a Tile. This is an optional consideration. The NTA result is equal to the number of marked values in your bitmap.

3.3.4 Compatible Locations

One interesting facet of Grid Alignment is that, for any given source portal, there are four destination portal locations that provide a Grid Offset of (0, 0). The first is just the equivalent Forward Tile of the Source Grid, but with an inverse surface direction. The other three are found by taking the 90, 180, and 270 degree rotation of this location. Refer Figure 3.9 for visualization of the four locations for a given Source Grid. These four destination portal locations all yield a Grid Offset of (0, 0). In other words, these locations are directly compatible in terms of Grid Offset.

An interesting observation stemming from this fact is that all Grid Offsets can be seen as the deviation from a compatible location. If you wanted to create a game that never had unreachable area, you could only place portal-able walls at compatible locations.

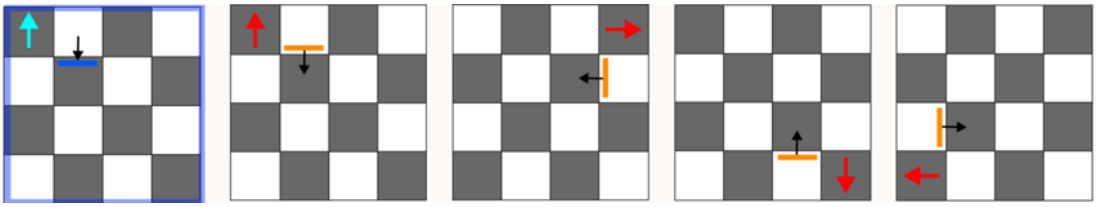


Figure 3.9: Four Compatible Locations for a Given Source Grid

3.3.5 NTA Algorithm - Example 1

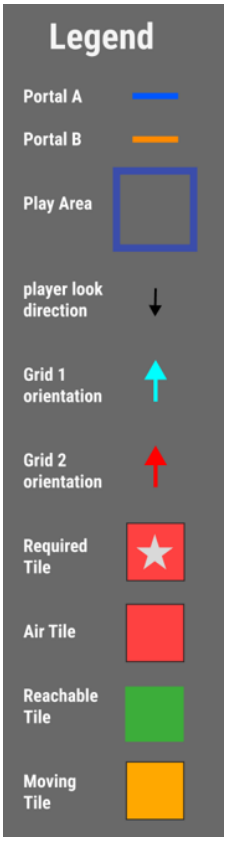


Figure 3.10: Legend for Grid Notation

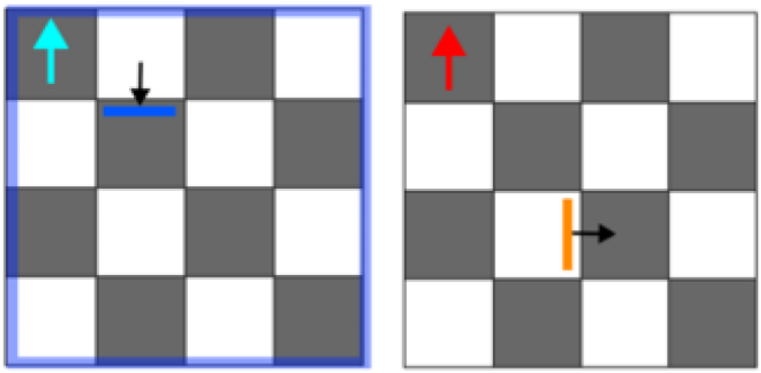


Figure 3.11: Source (left) and Destination (right) Grid example layouts. The Source Grid has a Play Area Offset of (0, 0) and a Source Portal on the North surface of Tile 6. The Destination Grid has a portal on the East surface of Tile 10

This first example demonstrates how the NTA Algorithm works step-by-step with Figure 3.11. In this example, the play area begins at an offset of (0, 0) to the Source Grid. The source portal is on Tile 6 facing North, while the destination portal is on Tile 10 facing East. The Forward Tile of the Source Grid is Tile 2, and Tile 11 for the Destination Grid.

Since the destination portal surface points East, we need to reorient the source to face West. This requires three 90-degree clockwise rotations of the Source Grid. After this transformation, the source Forward Tile becomes Tile 10. To follow the Grid Alignment steps, this should match with the Destination Grid's Portal Tile, which is Tile 11. Converting Tile 10 and Tile 11 to their (row, column) locations yields Tile 10: (2, 1) and Tile 11: (2, 2). In my implementation, this example yields a *Grid Offset* of (0, 1), indicating that the left column is invalid. Figure 3.12 provides a visual representation of this example.

To update the Play Area Offset correctly, it should undergo the same transformations. Three rotations of (0, 0) result in (0, 0). Adding the Grid Offset gives us (0, 1) which demonstrates that our base case produces the expected cal-

culation. As seen in Figure 3.12, the Play Area Offset to the Destination Grid is (0, 1).

Therefore, the play area encompasses all but four Tiles. The resulting NTA value is obtained as NTA = 16 - 4 = 12. If you need to account for Tiles blocked by obstacles, this is where you would consider subtracting applicable Tiles. In my case, I could subtract one for each Pillar Tile on the Destination Grid.

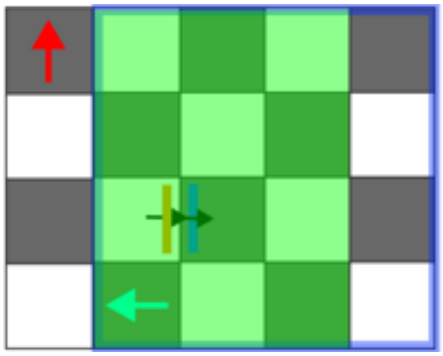


Figure 3.12: Overlap Solution to NTA Example 1. The result of portaling to this Destination Grid leaves 12 Tiles reachable by the player. The leftmost column of Tiles will be outside of their walkable play area and the Play Area Offset becomes (0, 1)

Finally, to provide a brute force assessment that demonstrates the soundness of this process, let us consider the local situation of each Grid. If the player is on Tile 2 and walks through the portal (which would place them on Tile 6) they can physically reach the following Tiles:

- 1 Tile to the Back
- 2 Tiles to the Front
- 2 Tiles to the Left
- 1 Tile to the Right

Upon being transported to Tile 11 of the Destination Grid, despite the change in the virtual world, the user's relative position to their play area remains the same. This implies that the reachability of Tiles must adhere to the same list. However, after examining Figure 3.11, we can observe the following Tiles to be "reachable":

- 2 Tile to the Back
- 1 Tiles to the Front
- 2 Tiles to the Left
- 1 Tile to the Right

Through this robust method, we can confirm that the leftmost column of Tiles is not reachable. In my implementation, I make this clear to the player by having the unreachable Tiles vanish from the virtual world. This addition aligns with the assessment by the NTA Algorithm, reinforcing the soundness of its results.

3.3.6 NTA Algorithm - Example 2

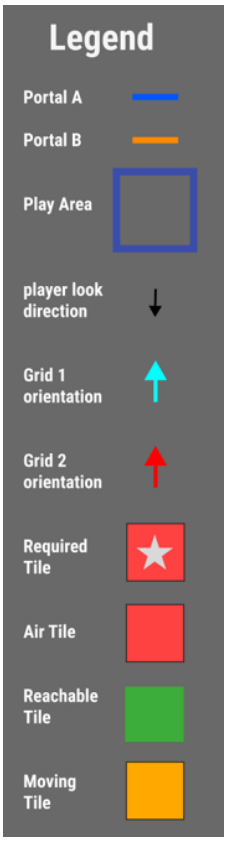


Figure 3.13: Legend for Grid Notation

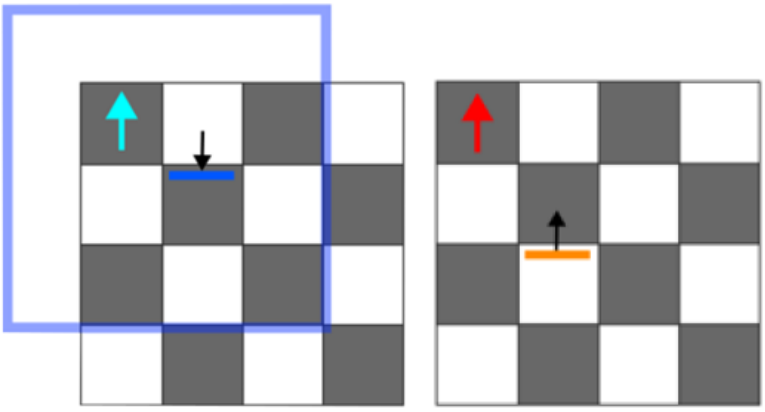


Figure 3.14: Source (left) and Destination (right) Grid example layouts. The Source Grid has a Play Area Offset of (-1, -1) and a Source Portal on the North surface of Tile 6. The Destination Grid has a portal on the North surface of Tile 10

The purpose of this second example is provide an example that begins with some Play Area Offset. This is both to demonstrate the importance of separating the Grid and Play Area Offset considerations and to show that this algorithm has infinite depth ability. Refer to Figure 3.14.

In this example, the Play Area Offset to the Source Grid begins at (-1, -1). The Source Grid otherwise has the same layout as in example 1. The source Forward Tile is Tile 2, while the destination Portal Tile is Tile 10. Both portals face North.

After rotating the Source Grid, the source Forward Tile becomes Tile 15. Converting the target alignment Tiles to (row, column) notation yields Tile 15: (3, 2) and Tile 10: (2, 1). In my implementation, this example yields a Grid Offset of (-1, -1).

To update the Play Area Offset, it must undergo two rotations as well. Therefore, it goes from (-1, -1) to (-1, 1) to (1, 1). After adding the Grid Offset of (-1, -1), we are left with a Play Area Offset of (0, 0) to the Destination Grid. This is illustrated in Figure 3.15.

An important observation from this example is that although there is some Grid Offset, all Tiles are reachable. This demonstrates why only the Play Area Offset is considered when determining reachable Tiles. For a different visualization, Figure 3.16 shows this in bitmap form. We can see how both the Grid Offset, labeled *PLAT OFFSET* here, and Play Area Offset would impact the Reachable Tiles Bitmap.

The final NTA solution for example 2 is 16, minus additional considerations if applicable.

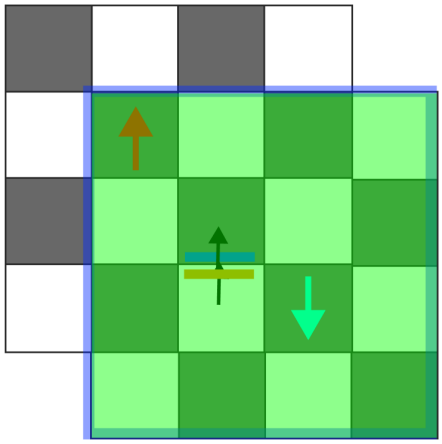


Figure 3.15: Overlap Solution to NTA Example 2. The result of portaling to this Destination Grid leaves 16 Tiles reachable by the player. Although the Grid Offset is (-1, -1), the resulting Play Area Offset is (0, 0), rendering all Tiles reachable

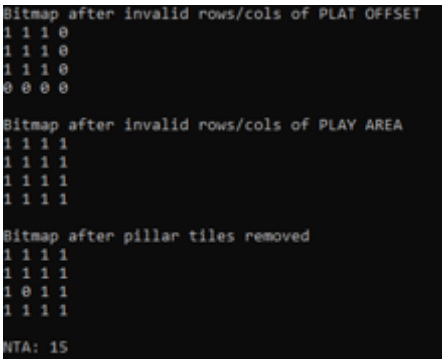


Figure 3.16: Reachable Tiles Bitmap Visual for NTA Example 2

3.4 Compatible Portal Location Algorithm

3.4.1 Overview

While the NTA Algorithm focuses on determining the available tiles within the game world, the Compatible Portal Location Algorithm complements this by identifying portal locations that are compatible with the world's structure and layout.

There are five key pieces of information that this algorithm relies on. First is the **Target game element**. The Target must contain a set of **Required Tiles**. These Required Tiles specify Tiles which must be reachable by the player once they reach the Target. This game element must have only one portalable wall in its layout.

The next essential component is the **Transition game element**. This refers to the Grid, or other relevant structure, where the algorithm will search for a compatible portal location. It is important to note that the Transition game element should be empty, devoid of any Pillar Tiles and Portalable walls.

Additionally, the algorithm requires a reference to a **Source game element**. For a linear game, this will be the game element prior to the Transition game element. However for nonlinear games, it can be any game element within range of the Transition game element. Finally, the **Play Area Offset** to the Source game element is required.

In my implementation of a nonlinear game-type, the Compatible Portal Location Algorithm is executed at runtime, whenever the player is on a Source that can reach a Transition. If the player changes their source portal location before moving to a Transition, the current compatible portal location is cleared and a new one is computed.

The use case I tested was for a Moving game element as the Target, and two Grids as the Source and Transition game elements. The Required Tiles were placed ahead of the Moving Tile to ensure the player has guaranteed walking space after using the game element.

At a high level, the Compatible Portal Location Algorithm is executed in X steps.

- Step 1: The player enters a Source game element that is in range of a Transition game element.
- Step 2: The NTA Algorithm is ran on each portalable wall on the Source with a random Tile surface on the Transition.
- Step 3: The NTA Algorithm is ran again, this time with the same random Tile surface and with the portalable wall on the Target.
- Step 4: If the NTA value is less than the Target's number of Required Tiles, return to step 2.
- Step 5: Using the resulting Reachable Tiles Bitmap from the second NTA calculation, determine if all Required Tiles are reachable
- Step 6: If all conditions were met, set the Compatible Portal Location on the Transition. Otherwise, return to step 2.

3.4.2 Algorithm Process

When the player first enters a relevant Source game element, the first step of the algorithm begins. To begin searching for a compatible portal location, we start by selecting a random Tile and random surface direction from the Transition. This randomly selected location is paired with each portalable surface available on the Source.

After running the NTA Algorithm on these two surfaces, we obtain the Play Area Offset to the Transition game element for the randomly selected portal location that we are testing.

Once the Play Area Offset to the Transition game element is known, we can plug the randomly selected portal location and the portalable surface of the

Target into the NTA Algorithm. Here the NTA value, Play Area Offset to the Target, and Reachable Tiles Bitmap is calculated. If the resulting NTA value is less than the number of Required Tiles, the algorithm moves on to the next iteration. Otherwise, if the Reachable Tiles Bitmap includes all Required Tiles, the algorithm returns True.

On a successful match, the algorithm can place a Compatible Portal Location on the Transition game element. In my implementation, with the use of Pillar Tiles, this means one portalable wall is placed with 3 Iron walls to create one Pillar Tile. This layout ensures that the Transition game element has a portal location which is compatible with their current Source Portal location and the Target game element.

3.4.3 Compatible Portal Location Algorithm - Example 1

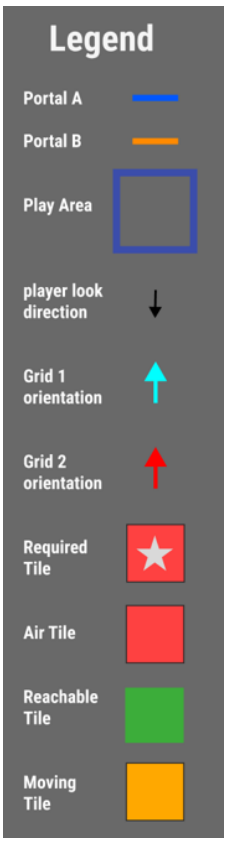


Figure 3.17: Legend for Grid Notation

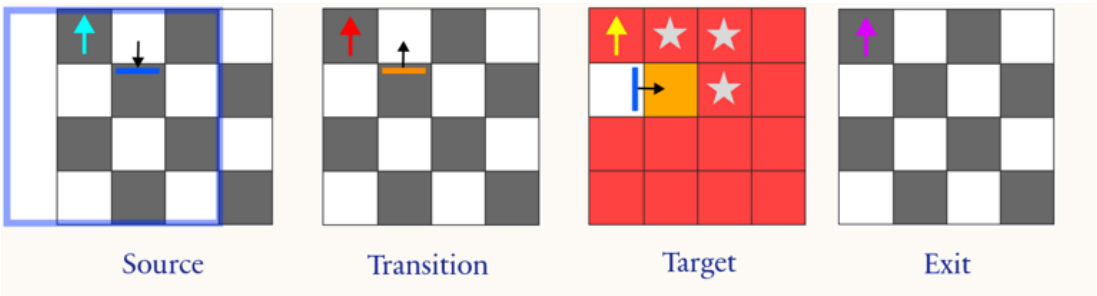


Figure 3.18: Compatible Portal Location Algorithm Grid Layouts

Figure 3.18 provides four Grids to demonstrate the Compatible Portal Location Algorithm. The Source Grid has a portal on the North surface of Tile 6 with a Play Area Offset of (0, -1). Coincidentally, the first randomly selected portal location for the Transition Grid is also the North surface of Tile 6. The Target Grid consists of 14 Air Tiles, a portal on the East surface of Tile 5, and a Moving Tile on Tile 6. Tiles 2, 3, and 7 have been marked as required. Lastly, there is an empty Exit Grid.

Using the Grid alignment process outlined in the NTA Algorithm section, the Source Grid undergoes two rotations and a translation of (-2, -1). This results in the Transition Grid having a Play Area Offset of (-2, 0). Figure 3.19 showcases the Grid Overlap on the left and the resulting Play Area Offset to the Transition on the right. It is worth noting that the resulting Transition Grid has a flipped player look direction, as its portal is used for exiting the Grid rather than entering.

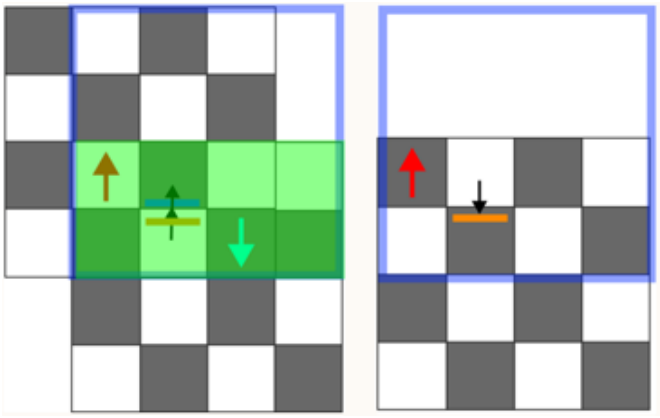


Figure 3.19: Source and Transition Grid Overlap for Compatible Portal Location 1. This Destination Portal would result in a Play Area Offset of (-2, 0), leaving the bottom two rows of Tiles unreachable

The next step is to align the Transition and Target Grids. Figure 3.20 displays the Grid Overlap on the left and the resulting Play Area Offset to the Target on the right. In both images, it is evident that only one of the three Required Tiles

will be reachable with the randomly selected portal location on the Transition Grid.

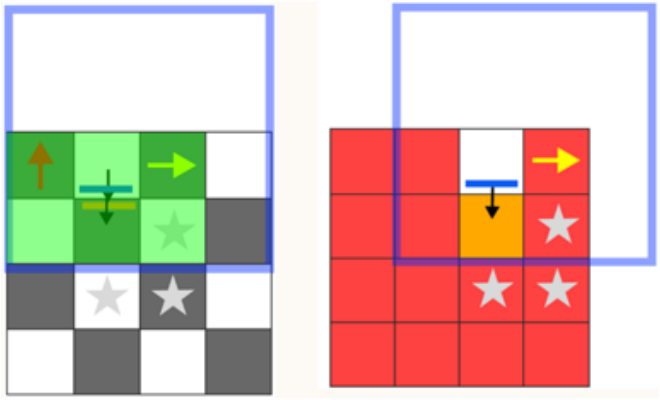


Figure 3.20: Transition and Target Grid Overlap for Compatible Portal Location 1. The Play Area Offset of (-2, 0) happens to interact with the given Target Grid (Moving Element) such that only 1 out of 3 Required Tiles would be reachable

The subsequent step of the Compatible Portal Location Algorithm is to locate a new random portal location. In this case, it is the North surface of Tile 15, as depicted in Figure 3.21.

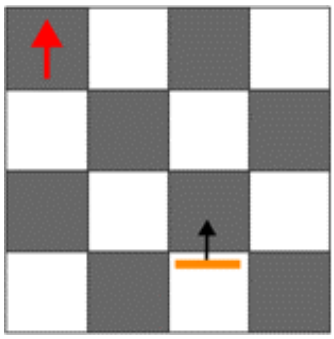


Figure 3.21: Transition Grid - Compatible Portal Location 2

After aligning the Source and new Transition Grid, the Play Area Offset becomes (1, 1) as shown in Figure 3.22. Once again, the right image showcases the resulting Transition Grid with its portal notation flipped.

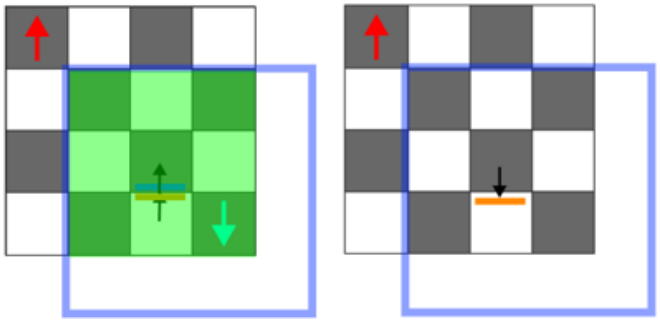


Figure 3.22: Source and Transition Grid Overlap for Compatible Portal Location 2. This Destination Portal would result in a Play Area Offset of (1, 1), leaving the top row and leftmost column of Tiles unreachable

Finally, Figure 3.23 illustrates the overlap between the Transition and Target Grids. With a Play Area Offset of (-1, -1) to the Target, all three Required Tiles are known to be reachable. At this point, the algorithm terminates and returns the successful location.

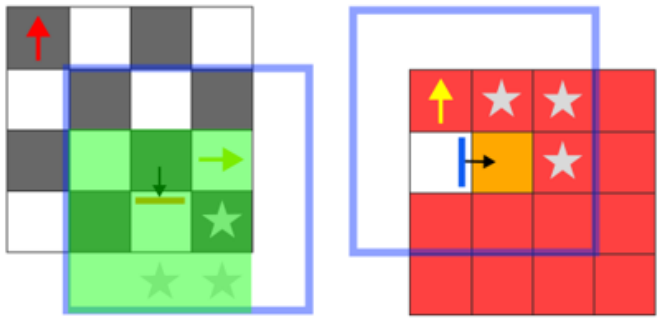


Figure 3.23: Transition and Target Grid Overlap for Compatible Portal Location 2. The Play Area Offset of (1, 1) happens to interact with the given Target Grid (Moving Element) such that only all 3 Required Tiles would be reachable

Chapter 4

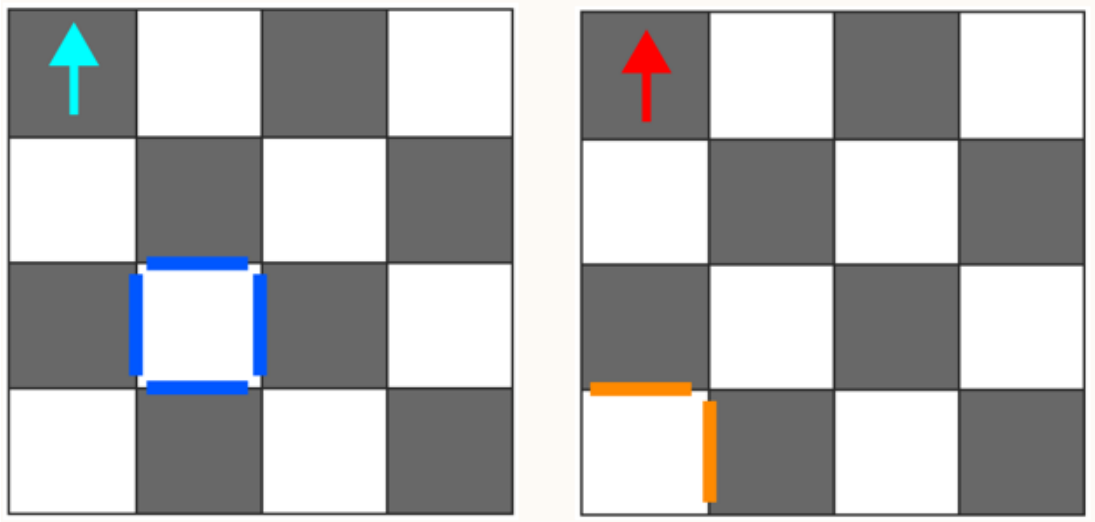
Results

4.1 NTA Algorithm

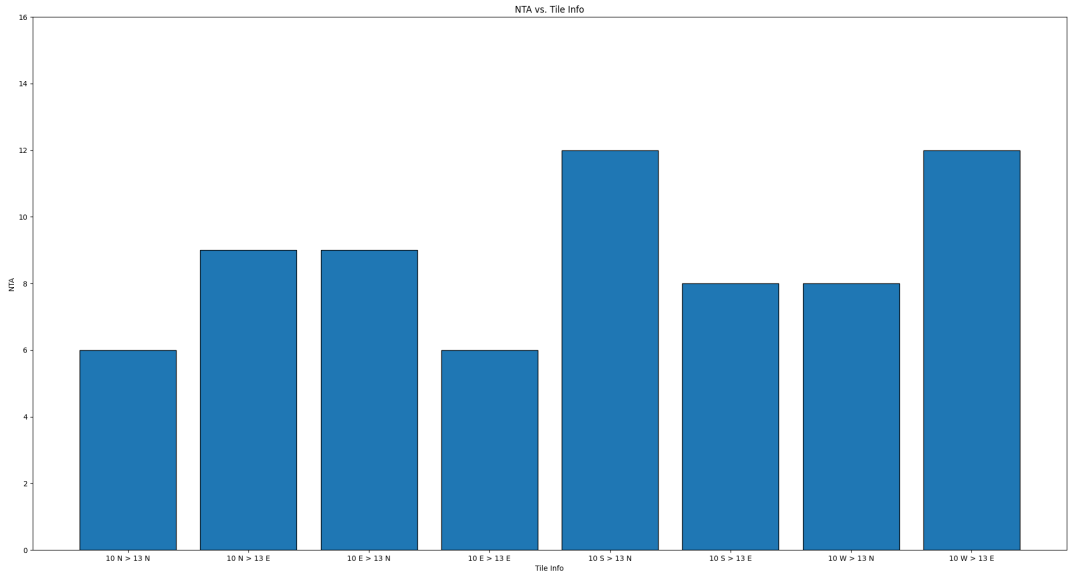
In this section, I present the results of the NTA Algorithm, which calculates how many tiles are available (NTA) for every combination of portal locations.

To visualize the NTA values, I created a bar graph that illustrates the NTA for each portal location combination. The x-axis represents the portal location combinations in the following format: [Source Grid Tile #] Orientation - [Destination Grid Tile #] Orientation. The y-axis represents the resulting NTA values for the destination Grid.

4.1.1 Examples



(a) NTA Algorithm Example 1 Layout. The Source Grid has a Pillar Tile on Tile 10 while the Destination Grid has one on Tile 13



(b) NTA for each Portal Combination between the two known Grids. A range of 0-16 is used to capture all possible NTA values

Figure 4.1: NTA Algorithm Example 1. Two Grids with one Pillar Tile per Grid. The Source Grid begins with a PAO of (0, 0)

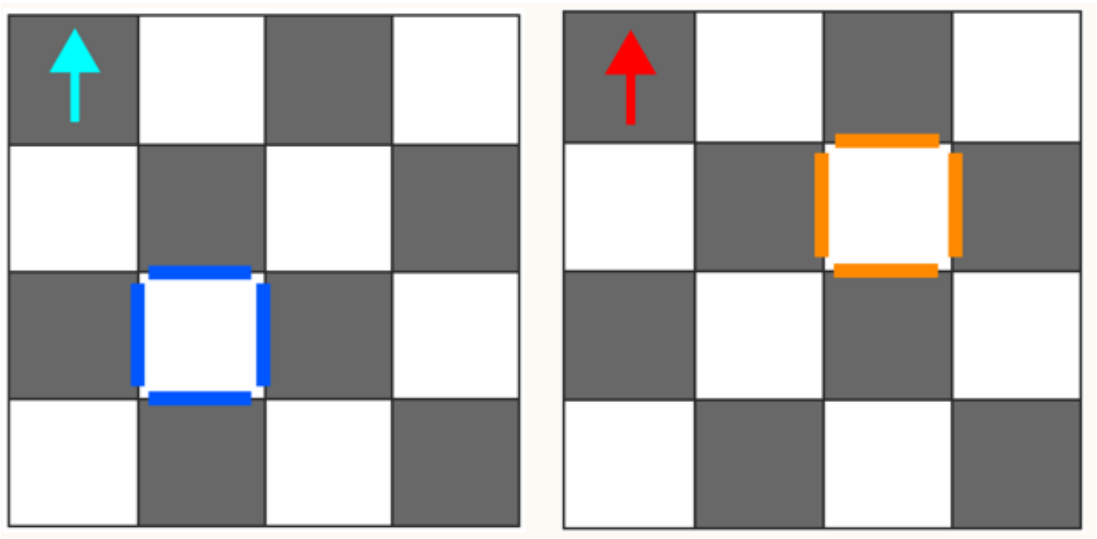
In Figure 4.1b, the resulting NTA data is shown for two Grids. The layout for both the Source and Destination Grid can be seen in Figure 4.1a. The Source Grid begins with a Play Area Offset of (0, 0) which will have minimal impact on the resulting NTA in this example.

Note 1: This example only has eight results since the Destination Grid only has two valid portal locations. A Pillar Tile placed in one of the corners of a Grid will always be limited to two orientations, as the other two would place the player off the virtual game area after portaling.

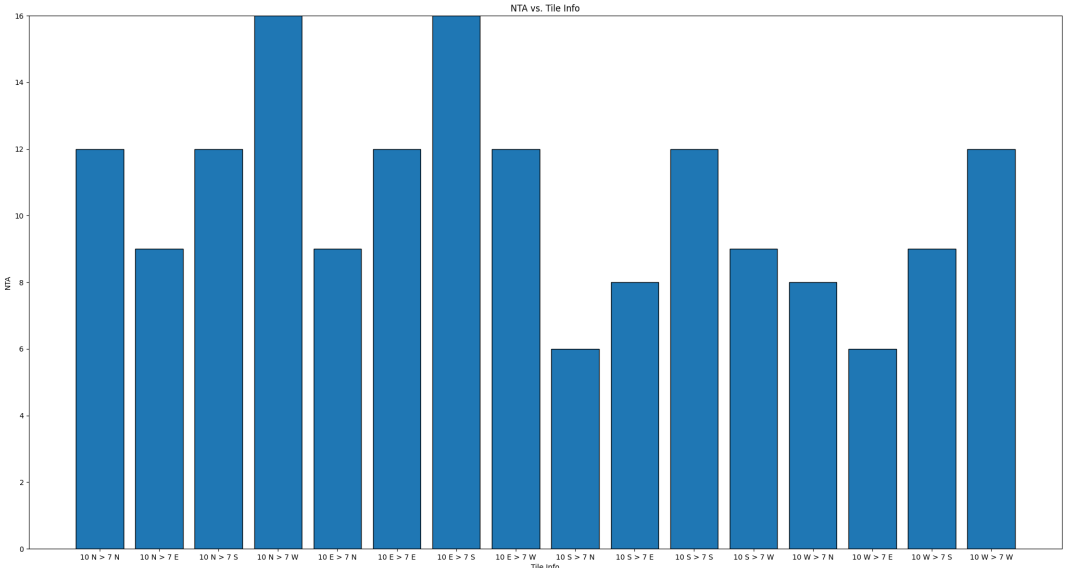
Observation 1: Upon examining Figure 4.1b, it is evident that each of the eight portal combinations provides a different number of reachable tiles (NTA). This variability in NTA indicates that the placement of portals can significantly impact the potential for exploration and movement within the virtual environment. Developers can leverage this insight to make informed decisions about portal options and strategically design for their desired gameplay experience.

Observation 2: In the case that developers opt to use Pillar Tiles, a design choice highlighted in this study, the use of Iron walls isolates portal locations and enhances the developer's agency over the virtual environment. For example, to enforce a greater range of freedom, Iron walls can be placed on portal locations that provide a low NTA value.

These observations demonstrate the practical usefulness of the NTA Algorithm for developers. By leveraging the algorithm's insights, informed design decisions can be made about a virtual world.



(a) NTA Algorithm Example 2 Layout. The Source Grid has a Pillar Tile on Tile 10 while the Destination Grid has one on Tile 7



(b) NTA for each Portal Combination between the two known Grids. A range of 0-16 is used to capture all possible NTA values

Figure 4.2: NTA Algorithm Example 2. Two Grids with one Pillar Tile per Grid. The Source Grid begins with a Play Area Offset of (0, 0)

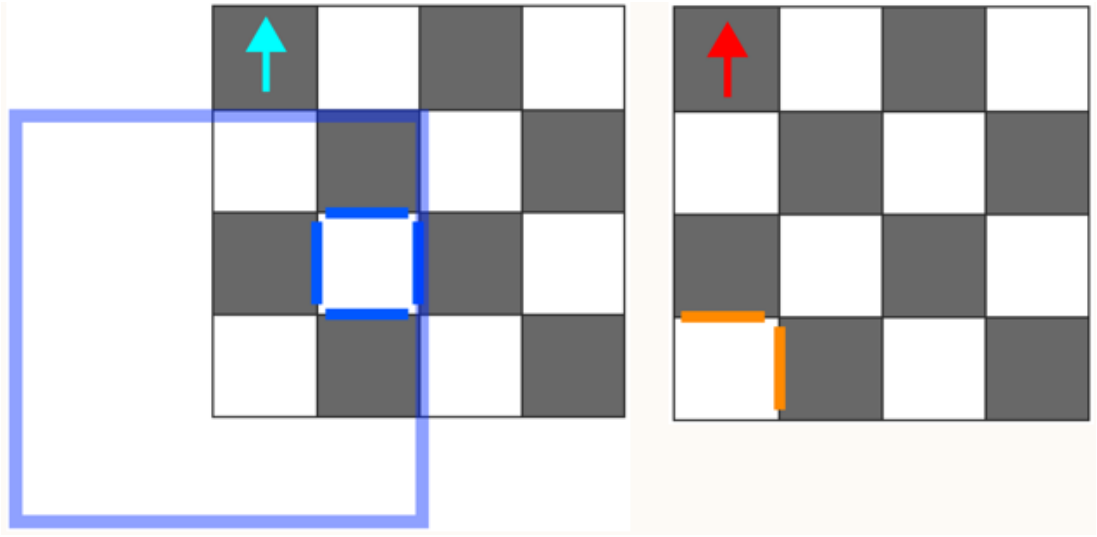
Note 2: This example has twice the number of results (16) as the previous example. The placement of Pillar Tiles in the middle of each Grid allows for all four orientations to be valid options. The Play Area Offset remains (0, 0), exerting minimal influence on the results.

Observation 3: Figure 4.2b showcases two data points with a perfect NTA value of 16. The first data point represents a Source Portal on the North face of Tile 10 and a Destination Portal on the West face of Tile 7. The second data point involves a Source Portal on the East face of Tile 10 and a Destination Portal on the South face of Tile 7. In both cases, the destination location is a Compatible Location to the source location.

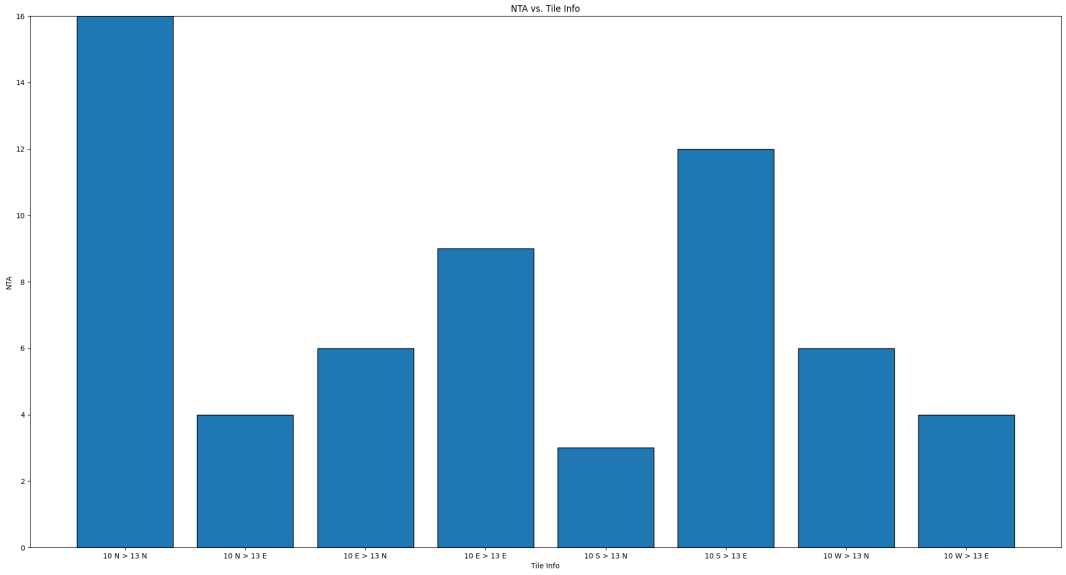
Observation 4: All NTA values are directly related to the destination location's offset from a related Compatible Location. As mentioned previously, there are four Compatible Locations to any source location. The *related* Compatible Location refers to the one with the same orientation as the destination portal location.

In Figure 4.2b, the lowest NTA value is 6 which corresponds to the combination of Tile 10 (South) and Tile 7 (North). The Compatible Location for the source in this case is Tile 14 (North), which yields an NTA value of 16. The offset between Tile 7 (North) and Tile 14 (North), is (-2, 1).

Comparatively, the first data point in Figure 4.2b, where the NTA value is calculated between Tile 10 (North) and Tile 7 (North). The related Compatible Location here is Tile 10 (North), and the offset of the destination to the Compatible Location is (-1, 0). This offset, being of lesser magnitude compared to the previous example, (-2, 1), results in an NTA value of 12, reflecting the relationship between the offset and the NTA value.



(a) NTA Algorithm Example 3 Layout. The Source Grid has a Pillar Tile on Tile 10 while the Destination Grid has one on Tile 13. The Source Grid also begins with a Play Area Offset of (1, -2)



(b) NTA for each Portal Combination between the two known Grids. A range of 0-16 is used to capture all possible NTA values

Figure 4.3: NTA Algorithm Example 3. Two Grids with one Pillar Tile per Grid. The Play Area Offset to the Source Grid is set to (1, -2) which renders its top row and two rightmost columns unreachable

Note 3: Example 3 has eight possible portal combinations due to the placement of one Pillar Tile in the middle and the other in a corner. The Play Area Offset is set to (1, -2), significantly influencing the results.

Observation 5: The Play Area Offset shifts compatible portal locations. In Figure 4.3b, there is one data point with a perfect NTA value of 16. This corresponds to the combination of Tile 10 (North) and Tile 13 (North).

By default, the related Compatible Location in this example would be Tile 11 (North). However, when considering the given Play Area Offset, we arrive at Tile 13 (North). This adjusted related Compatible Location explains why this data point yields an NTA value of 16.

Referring to Figure 4.4, the image on the left showcases what the result would be for a destination location of Tile 11 (North). While the offset between the two Grids is (0, 0), the shifted Play Area Offset renders 10 tiles unreachable.

Consequently, the right image in Figure 4.4 demonstrates how Tile 13 (North) happens to be the adjusted, related Compatible Location.

Note 4: The claims of Observation 4 are accentuated by Example 3. The data points with the lowest NTA values have the greatest deviation from their respective Compatible Locations.

In this example, the lowest NTA value is 3. This correlates to the data point observing the portal combination of Tile 10 (South) to Tile 13 (North). Accounting for the Play Area Offset, the adjusted, related Compatible Location is Tile 12 (North). The offset of the destination location to this location is (1, 3). This offset possesses a significantly greater magnitude than the offset of (0, 0) seen in Observation 5's example. These findings highlight the direct correlation between the offset magnitude and the corresponding NTA value.

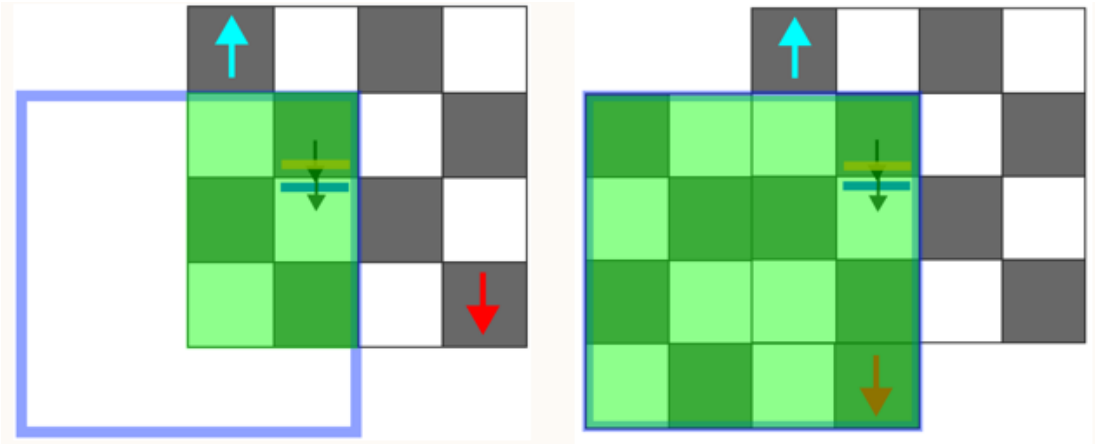


Figure 4.4: Grid Overlap Comparing a Default Compatible Location with a Compatible Location adjusted for the Play Area Offset

4.1.2 Generalization

In the context of the NTA Algorithm and the Grid-based data structure which utilizes Pillar Tiles, the entire problem space can be defined using known constraints.

Due to the inherent nature of a 4x4 Grid structure, there can only be three types of Pillar Tiles: Corner, Edge, and Middle, as demonstrated in Figure 5.1. Each Tile type has a distinct number of valid portals that it can support. Corner Tiles can support two valid portal locations, Edges can support three, and Middles can support four.

Three Pillar types results in six known combination types:

- Corner to Corner: Results in 4 portal combinations.
- Corner to Edge: Results in 6 portal combinations.
- Corner to Middle: Results in 8 portal combinations.
- Edge to Edge: Results in 9 portal combinations.

- Edge to Middle: Results in 12 portal combinations.
- Middle to Middle: Results in 16 portal combinations.

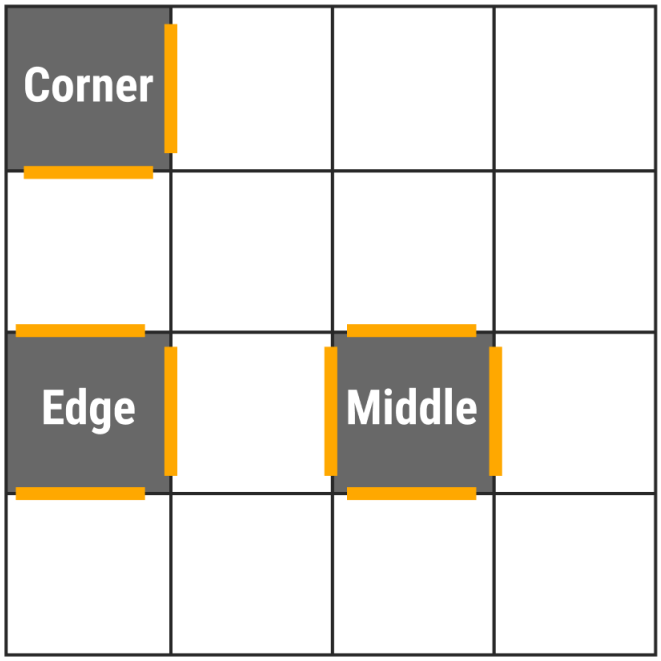


Figure 4.5: A Grid Template demonstrating Tile Types. The Yellow Bars indicate Valid Portal Locations

Figure 4.6 showcases each of the six combinations of Pillar Tile types. The x-axis lists all possible NTA values, 0 to 16, with each bar representing frequency.

Observation 1: Figure 4.6 shows that, with a Play Area Offset of (0, 0), NTA values of 0, 1, 5, 7, 10, 11, 13, 14, and 15 are not obtainable. Understanding that an NTA of 0 is not a possible outcome means the player will always have at least one Tile to stand on, for any portal location they choose. Careful consideration of valid portal location requirements, such as perimeter edges, is a major factor in this outcome.

Observation 2: Only half of the possible Pillar Type combinations can provide a perfect NTA value of 16. Similarly, only two can produce an NTA value of 2.

Observation 3: Known outcomes for Pillar combination types means broad suggestions/assumptions can be made. For example, we know that a Middle Pillar Tile type on a source Grid will produce an NTA value of 4 or greater, no matter the destination Pillar Tile type.

In the case where game design is centered around challenge and limited range of freedom, corner to corner Pillar types can be frequently utilized, as 2/4 portal combinations will lead to an NTA value of just 2. Further efforts to enforce this could be made by replacing the remaining 2 portal combinations with Iron walls.

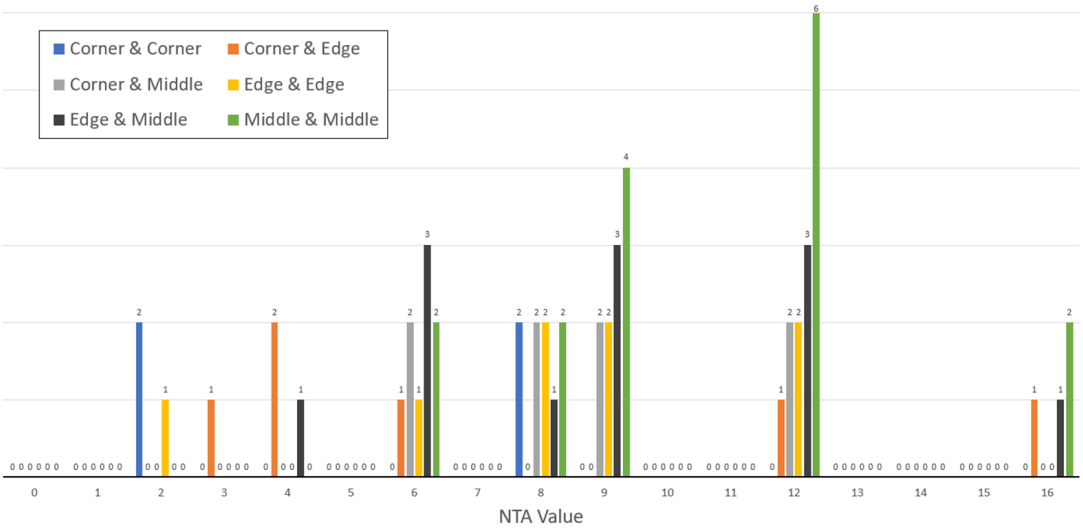


Figure 4.6: Bar Graph showing the NTA values for each Pillar permutation type. Frequency values for each NTA value are shown. The Play Area Offset is assumed to be (0, 0)

4.2 Compatible Portal Location Algorithm

The Compatible Portal Location Algorithm (CPL) takes a Source, Transition, and Target game element and determines a compatible portal location on the Transition. This compatibility guarantees that the required Tiles on the Target

will be reachable, taking into account the Play Area Offset of the Source, and how it transforms after portaling.

The CPL Algorithm can be ran in two modes: Ordered or Random. Ordered mode finds a Pillar Tile on the Source and tests a portal on one of the surfaces. Then it iterates over the Transition, testing Tile 1 (North), Tile 1 (East), and so on. However, it will skip *Invalid* locations. In this case, this pertains to perimeter edge considerations. Random takes every source to transition portal permutation and randomizes the order.

Figure 4.7 demonstrates that Random is vastly more efficient than Ordered. Additionally, Ordered always completes after 1, 49, or 97 location checks. Random can complete in any number of iterations. In this example, number of iterations was limited to 7 or less.

Observation 1: Despite differences in speed, the CPL Algorithm will always find a Compatible Location. This means custom game elements can be inserted anywhere into a virtual world of the game type I have proposed.

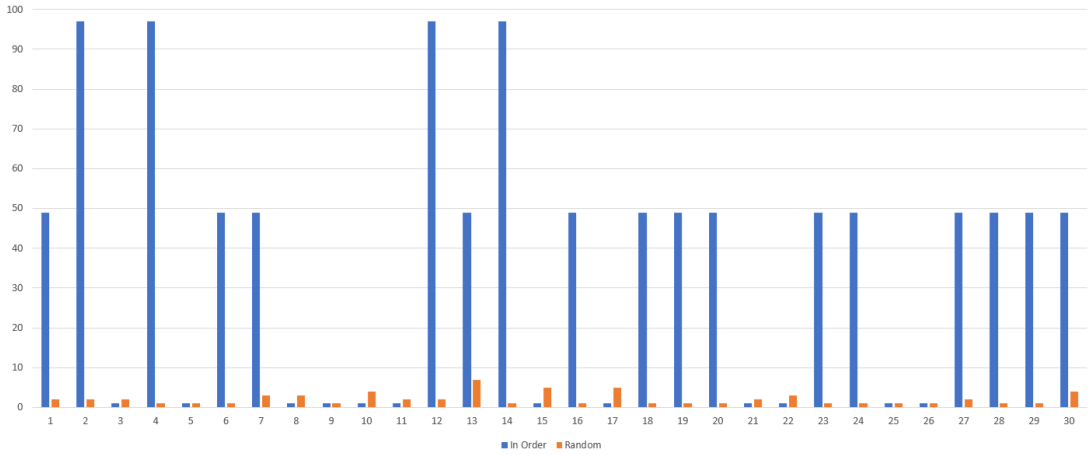


Figure 4.7: CPL tested in Random vs Ordered Mode. 30 test runs for each mode

Observation 2: The compatibility of a portal location permutation is en-

tirely dependent on the source portal location. In other words, if the North surface of a given Source Pillar Tile is finds a compatible location on the Transition, *all* portal location options on the Transition will be compatible with this source location. This relationship is demonstrated in Figure 4.8.

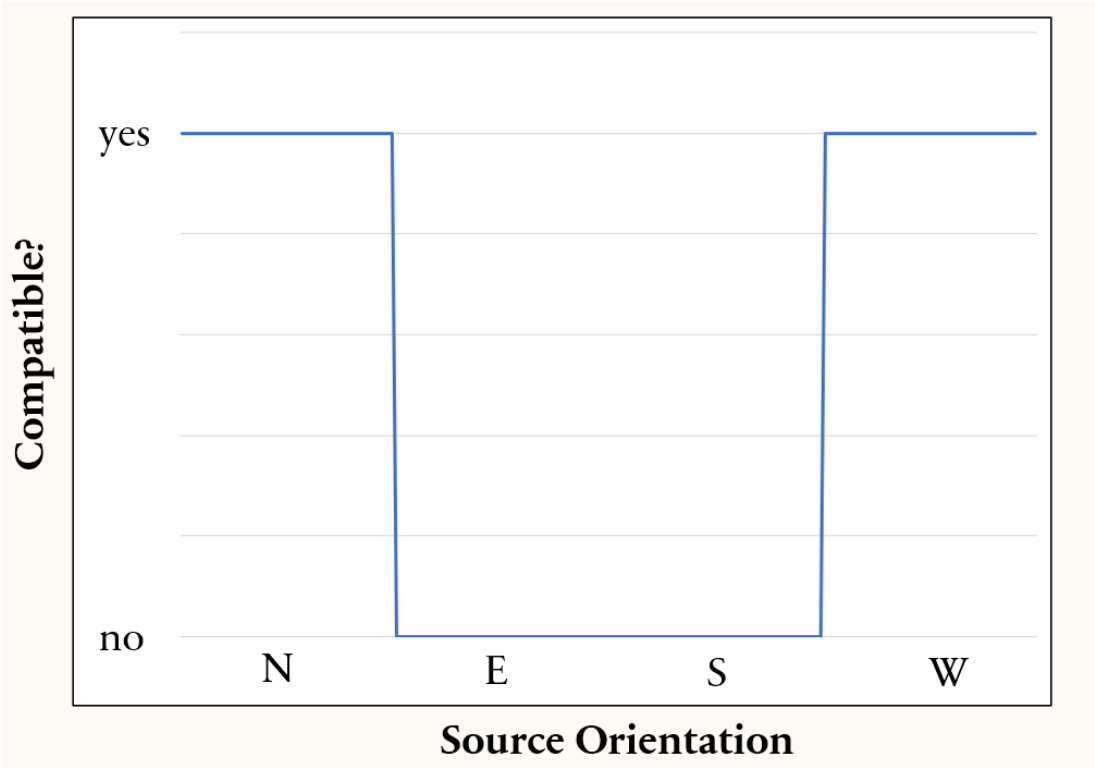


Figure 4.8: Results for every possible portal permutation for a random Source Grid and known Moving Element. There exists 48 permutations per source orientation

Chapter 5

Conclusion and Future Work

The research conducted in this work suggests a new game type that combines Room scale-based movement with a teleportation game mechanic (portals). With the use of the algorithms presented in this research, developers can make informed game design decisions to create virtual reality experiences based on this game type.

The NTA and CPL Algorithms can deliver feedback on the nature of the world built in this game type, or provide dynamic functionality for compatibility purposes. Developers can use these tools to prioritize large or small zones of freedom at different areas of a virtual world. Required Tiles allow for seamless compatibility to guarantee player access to designated parts of a Grid.

The focus on 4x4 Grids in this research has provided valuable insights into the practical application of the NTA and CPL Algorithms. Notably, the discovery of Compatible Locations and analysis of NTA value patterns have contributed to a deeper understanding of potential player experiences with my proposed game type. Pillar Tile type definitions provide useful, generalized information for less nuanced game design decisions, but can be expanded upon. In my research, I provided definitions and information surrounding the Corner, Middle, and Edge type. However, I believe there to be another type which only has one surface, as

opposed to two, three, and four, which appears when you have a non-zero Play Area Offset. Figure 5.1 shows how this unexplored Pillar Tile type can occur.

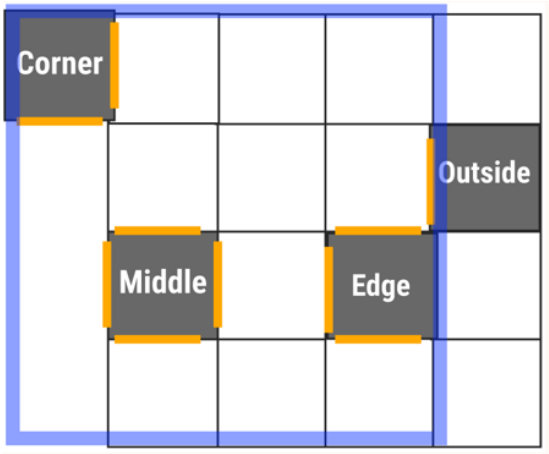


Figure 5.1: Demonstrates a fourth Pillar Tile type which occurs when there is a non-zero Play Area Offset

To expand the scope of future studies on this game type, it is important to investigate the effects of varying grid sizes, such as 5x5. The use of a different base Grid size, or of varying Grid sizes, will impact NTA and Compatible Location relationships with potential to create unexplored player experiences and freedom.

Furthermore, exploring alternative portal location constraints presents exciting opportunities. For instance, investigating the use of singular portalable walls, instead of Pillar Tiles. Or introducing free portal placement, based on player look rotation and position, instead of conforming to a Grid format. These variations offer unique challenges and creative potential with this game type.

Another aspect of research would be other transitional devices. This work relied on the use of portals, but the use of Non-euclidean Geometry as a mode of transporting the player could create new immersive experiences. New transitional devices would require a fresh analysis of NTA values and Play Area Offset relationships.

Bibliography

[1] Costas Boletsis. The new era of virtual reality locomotion: A systematic literature review of techniques and a proposed typology. *Multimodal Technologies and Interaction*, 1(4), 2017.

[2] Costas Boletsis and Jarl Erik Cedergren. VR locomotion in the new era of virtual reality: An empirical comparison of prevalent techniques. *Advances in Human-Computer Interaction*, 2019:1–15, April 2019.

[3] Costas Boletsis and Dimitra Chasanidou. A typology of virtual reality locomotion techniques. *Multimodal Technologies and Interaction*, 6(9), 2022.

[4] Evren Bozgeyikli, Andrew Raij, Srinivas Katkoori, and Rajiv Dubey. Point & teleport locomotion technique for virtual reality. CHI PLAY '16, page 205–216, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Fabio Buttussi and Luca Chittaro. Locomotion in place in virtual reality: A comparative evaluation of joystick, teleport, and leaning. *IEEE Transactions on Visualization and Computer Graphics*, 27(1):125–136, 2021.

[6] Kayla Davis, Taylor Hayase, Irene Humer, Brandon Woodard, and Christian Eckhardt. A quantitative analysis of redirected walking in virtual reality using saccadic eye movements. In *Advances in Visual Computing: 17th*

International Symposium, ISVC 2022, San Diego, CA, USA, October 3–5, 2022, Proceedings, Part II, pages 205–216. Springer, 2022.

[7] Malte Husung and Eike Langbehn. Of portals and orbs: An evaluation of scene transition techniques for virtual reality. In *Proceedings of Mensch Und Computer 2019*, MuC'19, page 245–254, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Eike Langbehn, Paul Lubos, Gerd Bruder, and Frank Steinicke. Application of redirected walking in room-scale vr. In *2017 IEEE Virtual Reality (VR)*, pages 449–450, 2017.

[9] Eike Langbehn, Paul Lubos, and Frank Steinicke. Evaluation of locomotion techniques for room-scale vr: Joystick, teleportation, and redirected walking. In *Proceedings of the Virtual Reality International Conference - Laval Virtual*, VRIC '18, New York, NY, USA, 2018. Association for Computing Machinery.

[10] Lisa Marie Prinz, Tintu Mathew, Simon Klüber, and Benjamin Weyers. An overview and analysis of publications on locomotion taxonomies. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 385–388, 2021.

[11] Frank Steinicke, Gerd Bruder, Jason Jerald, Harald Frenz, and Markus Lappe. Estimation of detection thresholds for redirected walking techniques. *IEEE Transactions on Visualization and Computer Graphics*, 16(1):17–27, 2010.

[12] Martin Usoh, Ernest Catena, Sima Arman, and Mel Slater. Using presence

questionnaires in reality. *Presence: Teleoperators and Virtual Environments*, 9(5):497–503, October 2000.