Tampere University of Applied Sciences

Conversion Between Square Grid and Hexagon Grid Playing Fields as a Game Mechanic

Veeti Karilainen

BACHELOR'S THESIS
March 2024

Degree Programme in Business Information Systems
Game Production

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma
Game Production

KARILAINEN, VEETI:
Neliöpohjaisen ja kuusikulmiopohjaisen pelialueen välinen konversio pelimekaniikkana

Opinnäytetyö 57 sivua
Maaliskuu 2024

Opinnäytetyön tavoitteena on tutkia mahdollisuutta muuttaa videopelissä käytettyä ruudukkopohjaa toisen muotoiseksi esim. Neliöruudukkopohja kuusikulmiopohjaksi. Lisäksi työssä esitellään erilaisia ja yleisimpiä peleissä käytettyjä ruudukkopohjia ja niiden ominaisuuksia. Työssä pyritään tutkimaan ja tasapainottamaan ruudukkopohjan muutosta pelimekaniikkana. Pelimekaniikkaa pyritään soveltamaan ensisijaisesti vuoropohjaisiin strategiapeleihin, joissa peli käydään ruudukkopohjaisella pelikentällä.

Erilaisia ruudukkoja, niiden merkitystä, ominaisuuksia sekä yleisyyttä tutkittiin ja arvioitiin erilaisten pelisuunnittelun, matematiikan sekä ruudukkopohjaisten pelien ammattilaisten julkaisemien materiaalien pohjalta. Ruudukkopohjan muodon muuttamista pelimekaniikkana, sen toimivuutta ja mahdollisia puutteita tutkittiin luomalla peliprototyyppi Unity-pelimoottorissa. Prototyypin avulla suunniteltiin mahdollisia rajoituksia ja muita pelimekaniikkoja sen käytön tasapainottamiseksi.

Opinnäytetyöprosessin perusteella voidaan todeta, että ruudukkopohjan muuttaminen toimii pelimekaniikkana, sillä se vaikuttaa merkittävästi pelin kulkuun. Ruudukkopohjan vaihdon tuottama hyöty riippuu paljon kontekstista, ja käytön yksipuolisuutta voidaan ehkäistä tarkasti suunnitelluilla rajoituksilla ja mekaniikoilla.

Asiasanat: Unity, pelisuunnittelu, videopeli ohjelmointi, neliöpohja, kuusikulmiopohja

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Production

Veeti Karilainen
Conversion Between Square Grid and Hexagon Grid Playing Fields as a Game Mechanic

Bachelor's thesis 57 pages
March 2024

The objectives of the thesis were to examine different, common grid layouts used in games and their characteristics, as well as to explore the possibility of changing the grid layout used in a videogame to a different shape, such as converting a square grid to a hexagon grid. This conversion was studied as a potential game mechanic, especially for turn-based strategy games, where the game is played on a grid-based playing field.

Different grid layouts' features and prevalence were researched and evaluated based on research published by various game design, mathematics, and grid-based game professionals. The conversion of the grid layout's shape as a game mechanic, its functionality, and potential shortcomings were studied by creating a game prototype in the Unity game engine.

Through the research and design of the mechanic, as well as the creation of the prototype, it can be concluded that changing the grid layout has a significant enough impact on the gameplay to justify its use as a game mechanic. Although a specific grid layout can provide the player with a significant advantage depending on the context, this can be balanced and mitigated with carefully designed restrictions and mechanics.

Key words: Unity, video game programming, game design, hexagon grid, square grid

CONTENTS

1 THE IMPORTANCE AND USE OF GRID IN VIDEOGAMES 5
2 DEFINITION AND SCIENCE OF GRIDS 7
2.1 Geometry and shape of grids 7
2.2 Tessellation 9
3 GRIDS IN VIDEOGAMES 10
3.1 Square grids 10
3.1.1 Strengths of a square grid 11
3.1.2 Weaknesses of a square grid 11
3.2 Hexagon grids 14
3.2.1 Strengths of a hexagon grid 21
3.2.2 Weaknesses of a hexagon grid 23
3.3 Triangle grids 24
4 CODE DEVELOPMENT OF GRIDS 28
4.1 Creation of a grid 28
4.2 Use of grids and pathfinding implementation 31
4.2.1 Pathfinding 31
4.2.2 Reading input 35
5 SYSTEM OF CONVERSION BETWEEN SQUARE AND HEXAGON GRIDS 38
5.1 Square grid to hexagon grid conversion and vice-versa 38
5.1.1 Theory of conversion study process 38
5.1.2 Implementation 39
5.2 Conversion system’s impact in-game 41
5.2.1 Gameplay impact of the conversion system 42
5.2.2 Usage of the conversion system as a game mechanic 45
6 CONCLUSIONS AND DISCUSSION 51
REFERENCES 54

1 THE IMPORTANCE AND USE OF GRID IN VIDEOGAMES

Grids are a popular and prominent tool for the creation of games, used for board games and video games of many different game genres (Patel 2006). Some of the most iconic and well-aged board games like chess and many more are played on a square grid, whereas the classic board game Agon consists of a grid made of hexagons. In both games, the game board is segmented into smaller areas of squares and hexagons respectively.

Grids are used for their simple and easy-to-digest visual presentation and use. They can be used to help visualize areas in maps, as well as be used for visualizing data within an area for a data map. In game design they have been used for their easy to grasp borders and areas, and for creating a confined and segmented play area. In a game like chess, it is simple to see the distinction between individual squares and understand distances between different squares. When a piece is governed by its own set of rules in a confined grid playing field, it is simpler to understand the effects and importance of different pieces.

Grids are an excellent solution for segmenting playing fields, for example for strategy games and for defining clear rules for movement on such a playing field. This is why their simple and effective design is and has been effective.

A game designer should consider the type of grid to use thoroughly when creating a game with a grid base. The clear geometrical differences of different shaped grid cells means that a developer should think before choosing a shape, so the number of neighbors for each cell and the number of sides they have as well as any other difference can be accounted for in game development.

This thesis work studies the definition, use, science and differences between different types of grids, and ultimately aims to create a system to see how different grid types can be smoothly converted into each other in a video game, without data loss or unnecessary distortion of the grid area. This system is developed with the pretense of being a core gameplay element in a turn-based strategy video game that is played on a grid.

This system would allow the player to have the ability to shape and convert the playing field from a square-based grid to a hex-based grid to be strategically more beneficial to them, and vice versa. This “switching” of core game elements that majorly define the core gameplay takes inspiration from other interactive media and video games that do similar switching between core game elements, such as most commonly switching between two-dimensional and three-dimensional gameplay areas. This system should have a sufficient impact on gameplay, as the differences of grids are further highlighted in gameplay in many different games, and having the power to switch between grid types should be a powerful mechanic for the player especially in turn-based strategy games.

2 DEFINITION AND SCIENCE OF GRIDS

Grids, defined by mathematical functions, segment surfaces for a clearer understanding of space and object locations. They play a crucial role in translating spatial information from maps. Grid shapes, sizes, and types vary based on the use case and data requirements, with squares and triangles being common choices. (Apte, Agarwadkar, Azmi, Inamdar 2013, 25.)

Grids being able to be constructed of different shapes means different shaped grid cells will have different properties and preferred use cases.

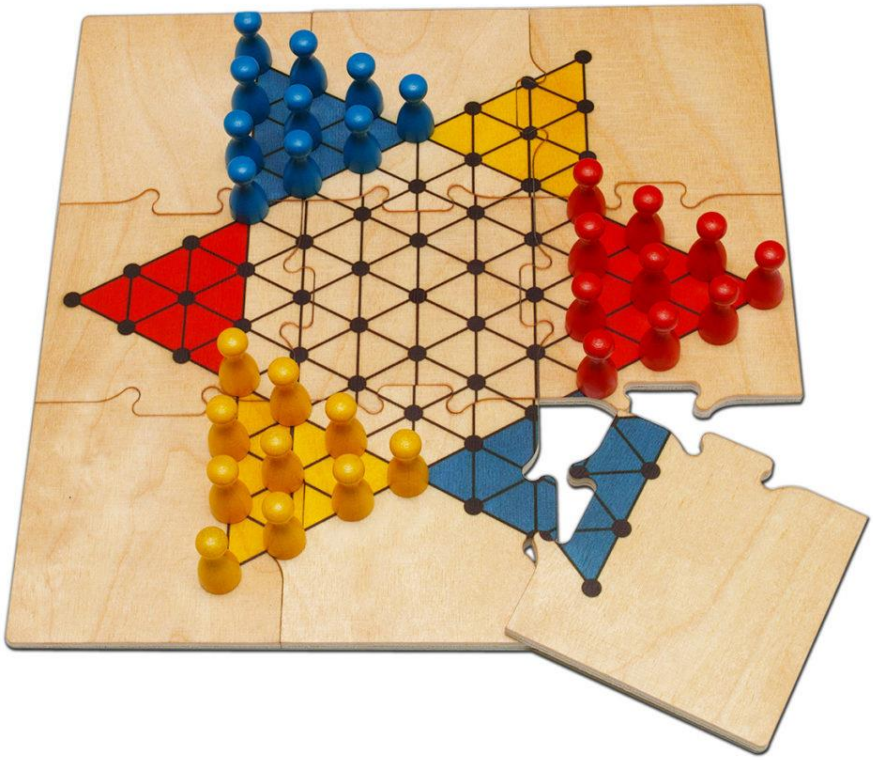
2.1 Geometry and shape of grids

Individual cells of grids are most commonly typical geometrical shapes, such as triangles, squares or hexagons. As a grid is a structure built from a repetition of a single simple shape, they are commonly only built from these three shapes. (Patel 2006.)

The individual cells' shape and geometry determine capabilities and potential of individual tiles. It is possible to easily understand functionalities and the relations of cells on a grid just based on the shape of its cells, as ruled by the geometrical features of the cell. These grid cell shapes have many attributes defined by how many sides the shape has, like the number of vertices, primary neighbors and secondary neighbors. Square cells, triangle cells and hexagon cells have 4, 3 and 6 sides respectively, and can at maximum have that many neighbors (Patel 2006).

By the mathematical definition of a triangle, each triangle grid cell has 3 vertices and can have up to 3 direct neighbors as triangles have 3 sides. Additionally, each vertex of the triangle is shared by a total of 6 triangles, 1 of the 6 being the triangle cell itself, 2 being direct neighbors and 3 of them being secondary neighbors. (Patel 2006.)

These attributes can be useful depending on the use case of the grid, especially when designing a game. For example: If a game permits a player to move from the center of the triangle, also known as its face, to the face of one of its direct neighbors, then we know that the player has the potential to make up to 3 different moves. On the other hand, if the game only allows the player to make moves between the vertices of a triangle, the player can make up to 6 distinct moves. In this case, the triangle grids movement options are reminiscent of that of the hexagon grids (Picture 1). This grid is only styled to look like a triangle grid but is functioning like a hexagon grid gameplay-wise due to each piece having 6 neighbors as movement options. (Boris 2021b.)



PICTURE 1. Chinese checkers, a game with a triangle grid where pieces are moved along the vertices (Boris 2021b).

Another key point of grids is the distance between center points of two neighboring grid cells. It is preferred that the distance between two center points of neighboring cells is universal in the entire grid, as cells having multiple different types of neighbors each of which are a different distance away complicates distance calculations.

When looking for the distance between a cell and one of its secondary neighbors, the distance is different than the distance to a directly neighboring cell in the case of square grids and triangle grid. Ryan Hill describes on their blog Horrible Pain (2016) how a square grid's secondary neighbors are a greater distance away than a square cells direct neighbors, and Phil Dutré (2018) defines that the triangles secondary neighbors are all a further distance away than its direct neighbors.

It is important to be aware of the possibilities offered by the structure of the grid cell when deciding on one. This helps be more efficient when working with the grid in question.

2.2 Tessellation

The previous chapters have only discussed three different shapes for the grid's cells, these being the square, hexagon and triangle. The reason for this is that these are some of the only possible common shapes that you can make a proper grid of by repeating the same shape. Filling a plane or a grid with repeating shapes without leaving any gaps is called a tessellation. (Pierce 2023.)

The regular tessellation is when only a single shape is repeated on a plane to fill it, without leaving any gaps in between the tiles. This is what square grids, hexagon grids and triangle grids are, and they are the only shapes that fit the description of a regular tessellation. This means no other single shape can repeat indefinitely to fill a plane without leaving gaps in between. (Pierce 2023.)

There are many other tessellations which differ in their internal rules for selecting the shapes to fill the plane with, for example by filling the plane by repeating multiple different shapes (Pierce 2023). There are also many different forms of tessellation besides the regular tessellation such as forms of semi-regular tessellations, wallpaper, and aperiodic tiling sets, but they are not commonly used in video game development and will be ignored for this study. Thus, only square, hexagon and triangle grids will be studied more thoroughly.

3 GRIDS IN VIDEOGAMES

Grids have been used in video games nearly as far back as video games have been made. The tile-based video game model was first established with the arcade game Galaxian by Namco in 1979 using 8x8 pixels and was the most popular with 8-bit and 16-bit video games until fading in popularity with the industry leaning more towards 3D graphics with the rise of 64-bit consoles in the late ‘90’s. Tile-based video games use a tileset to construct the playing field of individual, often square tiles. The result is a playing field built of individual square tiles laid out in a grid. This new technology for making games helped save space and pack more detail into games. Hexagonal tilesets were also created and used at the time, but to a much lesser extent, with only a handful of games of note. Though tile-based games made with tilemaps may have declined in popularity since developers got to use more powerful hardware, the creation of tile-based games was essential for the birth of grid-based videogames. (Wolf 2012, 173.)

3.1 Square grids

Square grids have been a popular solution in board games, as well as the most common grid type in video games (Patel 2006). In video games, the earliest use cases of square grids were heavily tied to the use of tilemaps for efficiency purposes.

Squares being the most popular choice of grids is not coincidental, with the grid type having many strengths. But it is not perfect, having many traits to be wary and critical of when designing a game, especially with grid-based movement systems.

3.1.1 Strengths of a square grid

The square grid's simple and easy design makes it have a low barrier of entry for game developers. This is partly due to how the square grid is so simple, easy to use, and how well the square maps to a computer screen. The square grid's locations work extremely well with the Cartesian coordinate system familiar to most, where each cell of the grid can easily be assigned a value on the X and Y axes of the grid. (Patel 2006.)

The square grid's cells each have 4 direct neighbors and 4 secondary neighbors. This can be a boon, as even Chess has mechanics unique solely to moving diagonally to secondary neighbors, such as the movement of rooks, or how pawns capture opposing pieces. Game designer Ryan Hill (2016) describes how the trait of having diagonal neighbors is an idea that feels natural solely for square grids. This total of 8 neighbors can be beneficial to games, as an increased number of neighbors can increase gameplay options.

Square grids are great for situations where the playing field portrays something in a straight line or right angle, such as walls or corridors. Lines and geometry can be drawn in a direct line to any of the eight directions from a cell. Game designer Jeremy Lennert (2018) makes a comment about how it is more convenient to do right 90-degree angles on a square grid than other grid types. This is great when the playing field has square shaped architecture, such as houses or buildings, especially if the architecture takes multiple cells. Lennert also comments on how the right angles of the square shape can be more efficient for manufacturing purposes or storage, especially in the context of board games.

3.1.2 Weaknesses of a square grid

Usually with square grids in turn-based videogames where units or the player can only move a fixed distance or number of cells per turn, the distance to direct neighbors is a single unit of measurement. But the distance to secondary neighbors is greater than the distance to primary neighbors.

This can be proven with the Pythagorean theorem

$$a^2 + b^2 = c^2 , \qquad (1)$$

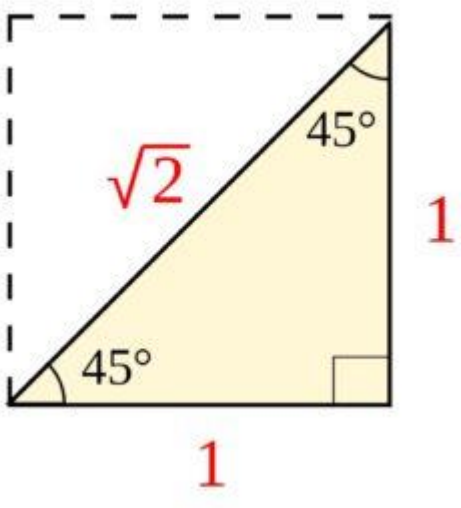
In which a and b are the legs of the triangle, and c is the hypotenuse of the triangle.

This distance can be visualized using a triangle (Picture 2) where the line drawn between the diagonal neighbor and the square form the hypotenuse, and the legs of the triangle are equal to the size of the square cell, in this case 1.

It is possible to get the distance to the secondary neighbor (Picture 2). In this case both sides *a* and *b* are of 1 unit of measurement in length and represent a horizontal and a vertical step on the grid, leaving the hypotenuse *c*, in this case the direct distance to the secondary neighbor. We can solve *c* using the formula (1)

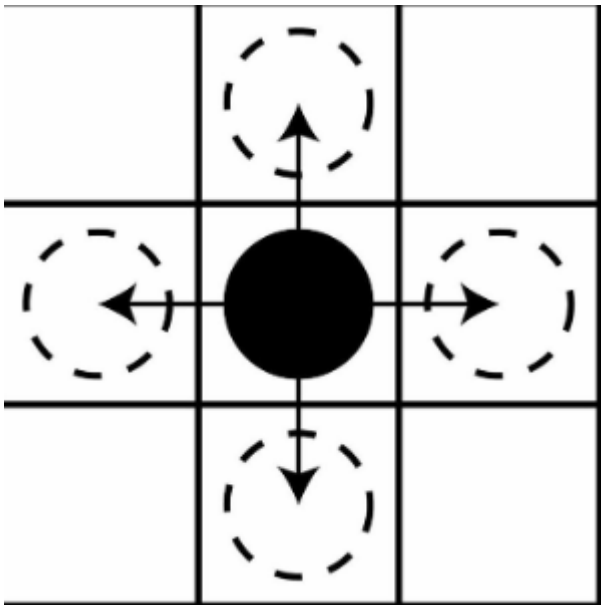
$$c = \sqrt{a + b} = \sqrt{1 + 1} = \sqrt{2}.$$

The distance to a secondary neighbor is in this case the square root of two, being roughly 1.41, compared to the distance of 1 to a direct neighbor.



PICTURE 2. Example showing the distance to a square cell's diagonal neighbor (Toppr n.d).

To avoid this inconsistency, in turn-based video games on a square grid, movement is typically only executed to direct neighbors either directly in horizontal or vertical directions. Such movement along only direct neighbors where steps are only carried out over the sides of a cell, in the case of the square grid only along the X and Y axes, is known as orthogonal movement (Picture 3) (Bond n.d).



PICTURE 3. How orthogonal movement works on a square grid (Bond n.d).

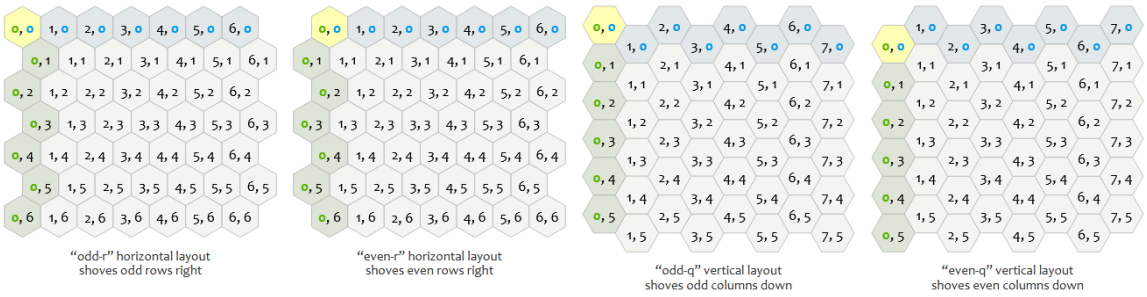
This movement system in square grid-based games is popular and, rather than addressing, ignores the inconsistency of distances between direct and secondary neighbors. Depending on if diagonal movement is allowed or not, movement to the secondary neighbors is either cheaper or more expensive than it should be. Either the player character gets to move directly to the secondary neighbor cell, and most often gets to move 1.41 distance for the same cost as moving 1 distance to a direct neighbor. This makes corner-to-corner movement cheaper, as the player gets to travel a greater in game distance for a movement cost. (Hill 2016.)

This is why many games on a square grid get rid of corner-to-corner movement altogether by only permitting orthogonal movement, but this does not solve the issue. Moving to a secondary neighbor would take moving 2 spaces to reach it. This makes corner-to-corner movement more expensive, since the actual distance of 1.41 is reached with 2 units of movement (Hill 2016).

3.2 Hexagon grids

The hexagon is a very strong shape that appears frequently in nature due to its efficient shape. Examples of hexagons in nature would be honeycombs, snowflakes as well as many cyclical chemical compounds. (Baugh 2022.)

Coordinates in a hexagon grid can be structured in many ways. The most common way of doing so is with the offset coordinate system. This method has variations (Picture 4) depending on if the hexagons are vertically or horizontally tiled, and whether the odd or even rows are offset. (Amit Patel 2013.)



PICTURE 4. Variations of offset coordinates in a hexagon grid (Amit Patel 2013).

Other methods for a hexagon grid's coordinates include cube coordinates, axial coordinates, which is a similar scheme to cube coordinates as well as the doubled coordinates (Amit Patel, 2013).

With a cube coordinate system, the coordinates are tracked with a three-axis model, shown as the q, r and s axes (Picture 5). In the cube coordinate scheme, a hexagon grid is considered like a slice taken out of a cube grid on the plane defined by the formula

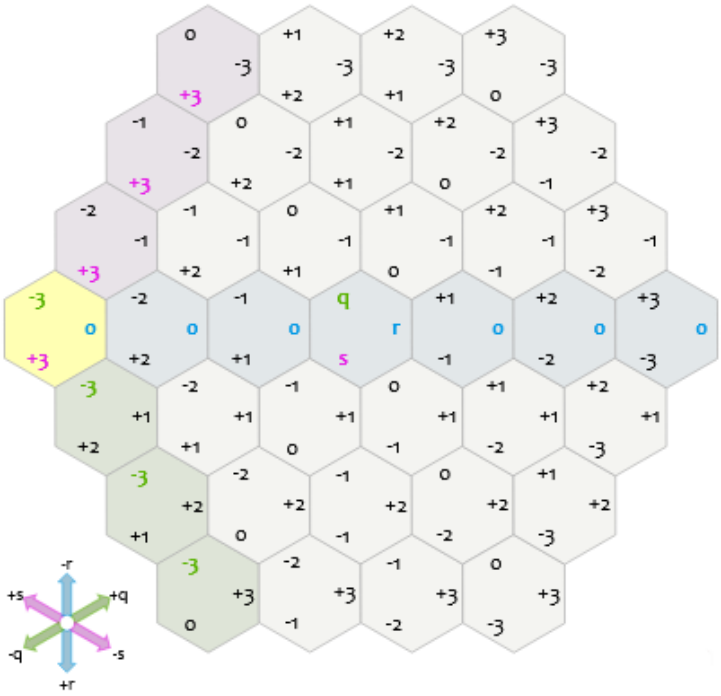
$$x + y + z = 0, \qquad (2)$$

where x, y and z are the coordinate values on each axis.

In the context of the grid this formula (2) is referred to as

$$q + r + s = 0, \qquad (3)$$

where q, r, and s replace x, y and z.



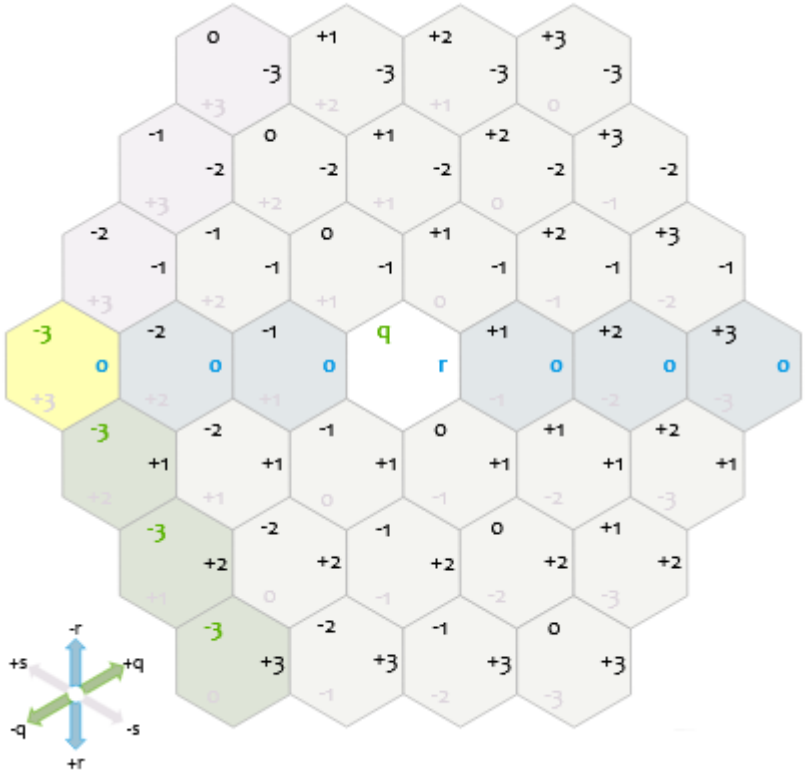
PICTURE 5. Cube coordinates in a hexagon grid (Amit Patel, 2013).

This perspective of three axes, and three dimensions, adapted to a hexagon grid allows for the adaptation of 3D cartesian coordinate algorithms, as hexagon grids have three primary axes, offering an elegant symmetry. Cube coordinates are beneficial for hex grid algorithms because they support standard vector operations such as addition, subtraction, multiplication and division, unlike offset coordinates. Algorithms for distances, rotation, reflection, and line drawing, as well as conversions to and from screen coordinates, can be adapted from 3D cartesian coordinates to hex grids. Each hexagon on the grid corresponds to a cube coordinate, and each direction on the hex grid is a combination of two cube grid directions. The constraint that the sum of the cube coordinates must be maintained according to formula (3) ensures a unique, canonical coordinate for each hexagon. The sum of any hexagon cell's coordinates must therefore be 0. (Amit Patel, 2013.)

The axial coordinate system is essentially the same as the cube coordinates, but with only two axes being tracked (Picture 6). The s-axis is not tracked with this method but since the constraint of the formula (3) still applies, s can still be calculated by shaping the formula (3) into

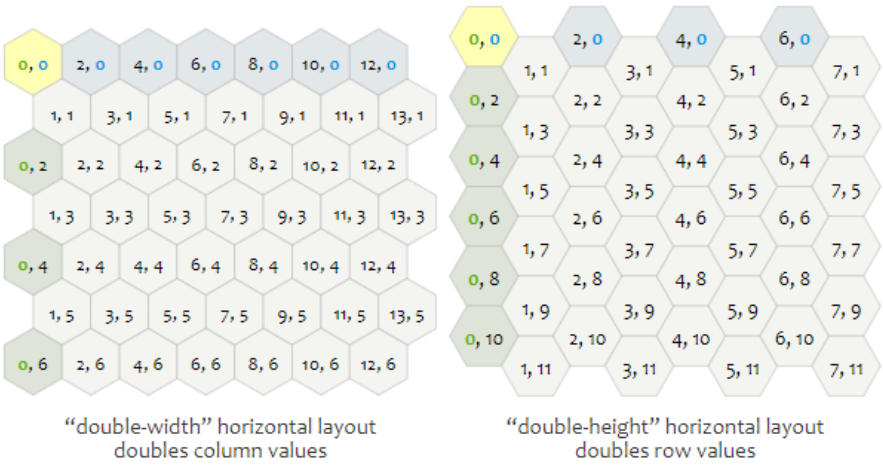
$$s = -q - r \qquad (4).$$

The cube and axial coordinate systems enable addition, subtraction, multiplication, and division with hex coordinates. This makes algorithms simpler for these coordinate systems, as the offset coordinate system is unable to use them. (Amit Patel 2013.)



PICTURE 6. Axial coordinate system in a hexagon grid (Patel 2013).

Doubled coordinates (Picture 7) is a system that is a variant of the offset coordinate system which differs from it to simplify implementing algorithms. Rather than alternating, either the vertical or horizontal step size is doubled. The horizontal layout increases the column by 2 each cell; in the vertical layout it increases the row by 2 each hex cell.

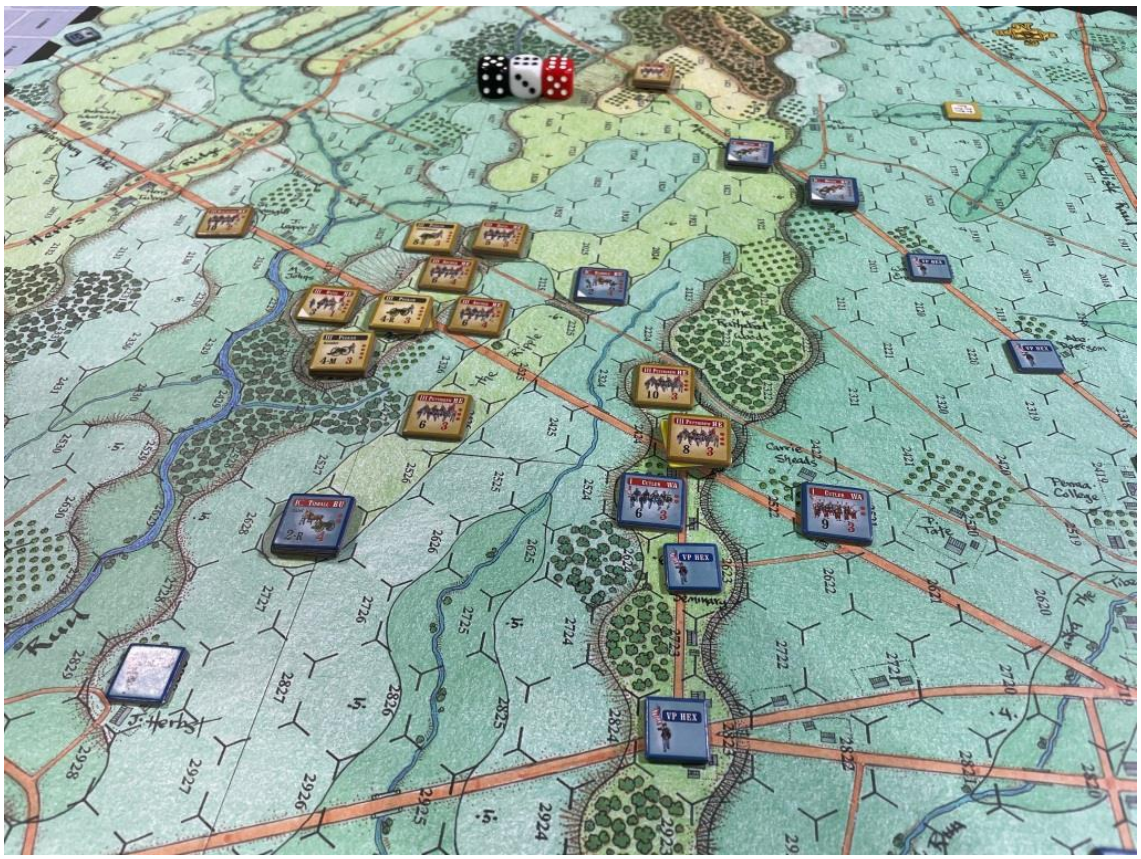


PICTURE 7. A doubled coordinate system in a hexagon grid (Patel 2013).

It is worth noting that there is uncertainty around the official name of the doubled coordinates scheme. There are many ways of calculating coordinates on a hexagon grid, and according to Patel (2013) it is also possible to have variations and completely different systems for coordinates on a hexagon grid. This thesis will primarily focus on the offset coordinates, as it is the most akin to the typical square grid's coordinates.

One of the hexagon grid's first uses in board games is as far back as Agon in the 18th century France (Tulleken 2014).

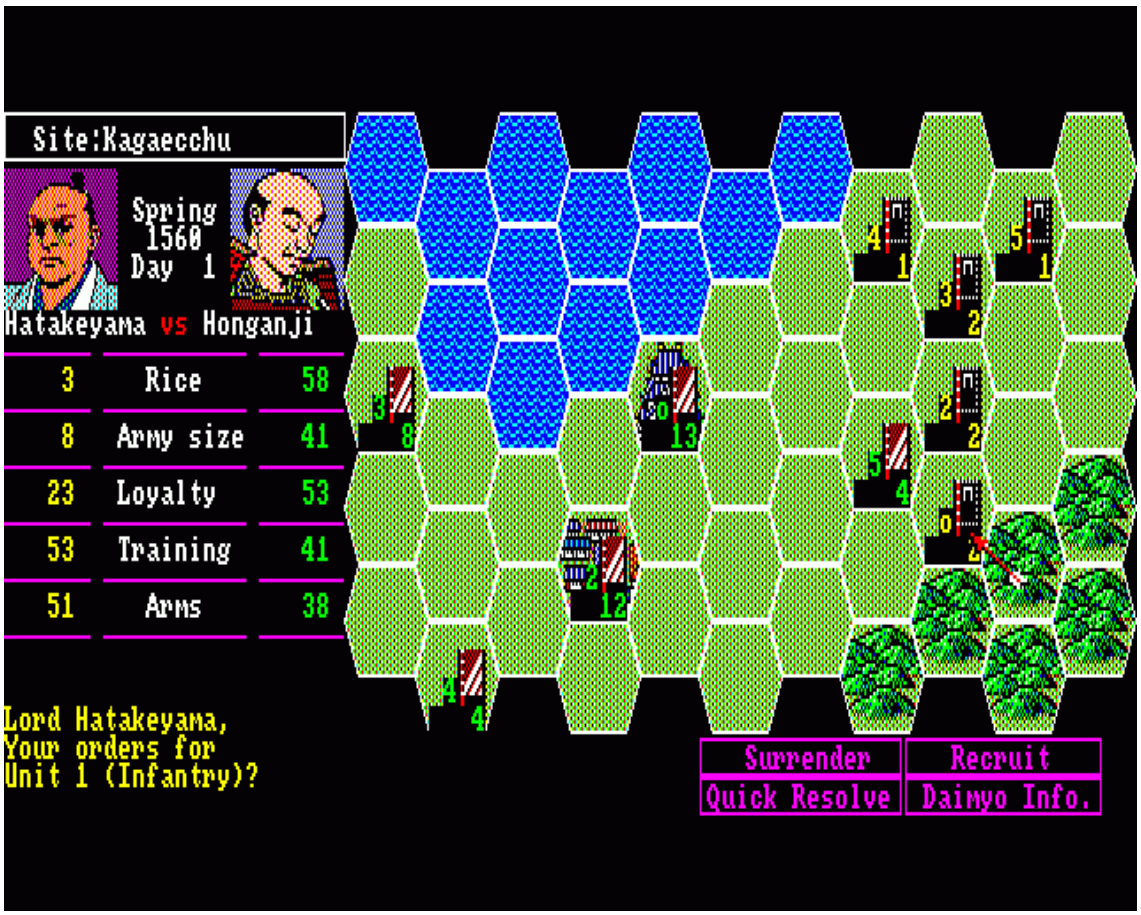
Other early uses of hexagons in games include the second edition of Avalon Hill game company's Gettysburg (Picture 8). The hexagon grid became popular in war games when the company's owner Charles Roberts introduced hexagons to commercial wargaming in the early 1950's. (Sidran 2018.)



PICTURE 8. The use of a hexagon grid in the board game Gettysburg (Grant 2023).

One of the earliest, if not the earliest, example of a hexagon grid used in video-games would be the turn-based strategy role playing game Nobunaga's Ambition, released in 1983 for personal computers and developed by Koei (Strategywiki n.d).

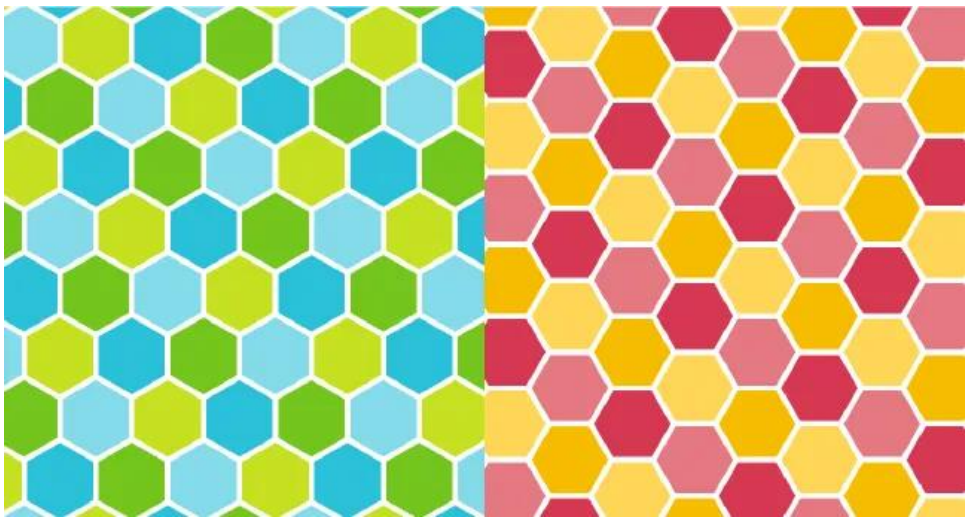
It is highly likely this is the first video game ever using a hexagon grid, but no concrete statement or proof proving or rebuking the fact was found. If it was the first video game to use hexagon, it is possible it had a big influence on popularizing the use of hexagon grids in video games. The game's grid (Picture 9) is highly reminiscent of today's hexagon grid-based video games.



PICTURE 9. Nobunaga‘s ambition released in Japan in 1983 was one of the first video games using a hex grid (Tobin 2020).

An additional trait of the hex grid is that its hexes all have the same direction that its developer has chosen. Put simply, the cells of a hexagon grid can either be with a vertex pointing up, or a side pointing up. And since the rows or columns of a hex grid can repeat in a pattern where every other one is offset, either every odd or even row will be offset.

Game design blogger Herman Tulleken (2014) talks about the impact of the choice between horizontal hexagon tiling and vertical hexagon tiling (Picture 10) by sharing helpful comments they found on gamedev.stackexchange.com. With horizontal tiling, they refer to a hexagon grid where the hexagons are positioned so that a vertex of the hexagon is facing up, creating a straight row of tiles horizontally. Vertical tiling refers to hexagons that have a side facing up, creating a straight vertical column of tiles. (Tulleken 2014.)



PICTURE 10. Examples of horizontal tiling on the left and vertical tiling on the right (Tulleken 2014).

The effects this choice has are related to gameplay design as well as visual design. A point that Tulleken (2014) brings up for gameplay is how the different methods of tiling the hexagon grid can have different optimal keyboard layouts for movement. Vertical hexes can adapt a movement scheme using the Q, W, E, A, S and D computer keys well, and horizontal hexes a movement scheme using the W, E, A, D, Z and X computer keys. Especially the W, E, A, D, Z and X layout of the horizontal tiling matches the typical layout of the keyboard well, with the S-key being the current cell. (Tulleken 2014.)

Tulleken (2014) explains how the choice between horizontal hexagon tiling and vertical hexagon tiling has different effects and potential on the side of visual design. The horizontal tiling is better for 3D or isometric perspective, where the bottom row is closer, and the top row is further above. “In this layout, high sprites won't obscure the center of the tile behind it, but just the edges” states Tulleken, and due to this states that vertical tiling could be better for a top-down view instead.

Tulleken’s comments also mentions how the vertical tiling can be better for the sake of how walls are presented, since vertically running walls can seem clunky with the horizontal tiling and you would not be able to see the details on the walls and concludes by saying how vertical tiling will result in a better look if

simulating bird's eye perspective, working with pixel art, or if the tiles are stretched sideways for depth. (Tulleken 2014.)

Overall, the hexagon grid has not been used quite as much as the square grid in video games, but this is by no means because it would be inferior. The hexagon grid offers many things the square grid does not, though it has its own flaws.

3.2.1 Strengths of a hexagon grid

People praise the hexagon grid for the more natural feel of having more orthogonal movement directions from each cell, as well as it being excellent for data mapping. One reason for this is that it is much more akin to a circle than a square is, which helps it appear less rigid and confined as the square, as well as make the flow of movement more fluid.

The hexagon has multiple strengths due to the shape's circularity. It has the advantage of spacing out each constituent hexagon evenly from its neighbors. Any given point inside a hexagon is closer to the center of that hexagon than any given point in an equal-area square or triangle would be, due to the square and triangle having more acute angles. Hexagons have the optimal efficiency for enclosure in comparison to other shapes that tessellate regularly, making it the best choice for tasks where both efficiency and ability to tessellate are required, because the hexagon is the closest to the ideal efficiency for enclosure of an area which is ultimately, a circle. (Graham 2010.)

Data scientist Matt Strimas-Mackey (2020) explains the benefits of the hexagon grid being the closest shape to a circle that can tessellate a plane regularly. Hexagonal grids excel in minimizing edge effects due to their low perimeter-to-area ratio, making them practical for various applications. This efficiency can be seen in the construction of beehives, where hexagonal honeycomb arrangements minimize material use while creating a lattice of cells with a specified volume. Hexagonal grids also stand out for their uniformity in neighboring cells, with each hexagon having six identical neighbors, sharing equal-length sides.

This is a notable contrast to square grids, which have two classes of neighbors—those in cardinal directions sharing an edge and those in diagonal directions sharing a vertex. Additionally, the distances between centroids are consistent for all hexagonal neighbors. When dealing with large areas where the curvature of the earth is significant, hexagons provide a superior fit compared to squares. This adaptability to curvature is exemplified in the construction of soccer balls, which utilize hexagonal panels for this reason. (Strimas-Mackey 2020.)

A good example of the optimal form of the hexagon would be honeycombs formed by honeybees. The honeybees optimize the usage of wax by forming the honeycombs into hexagons, due to the simple fact that you can fit the highest number of units into a plane if they are hexagon shaped, without having wasted space while having as small of a perimeter as possible (Baugh 2022).

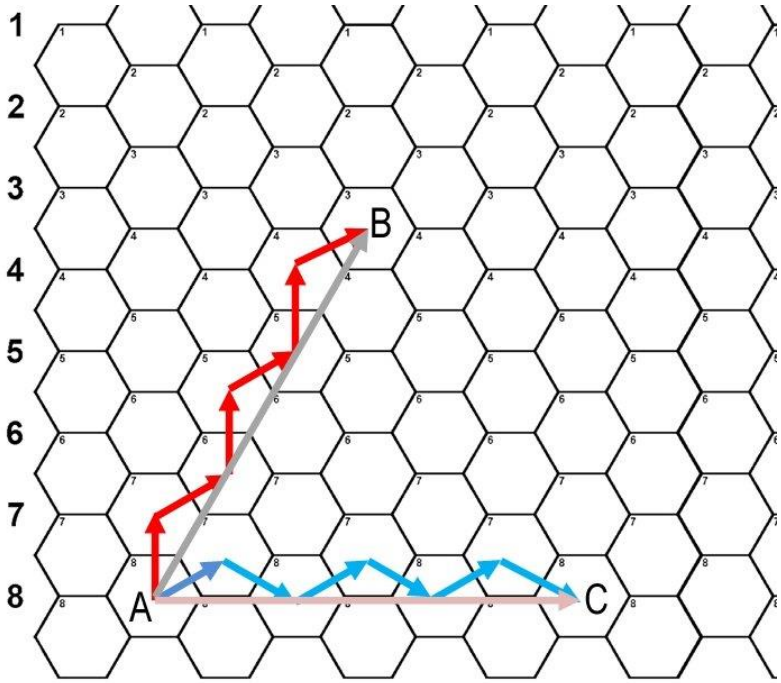
Also, like how Strimas-Mickey (2020) mentioned, a hexagon grid’s neighbors are all identical, with it only having direct neighbors, and no secondary neighbors that would share only a vertex. This means the hexagon-based grid has the advantage that its neighbors are all an identical distance away from each other. Thanks to this, orthogonal movement to all the neighbors of a hexagon shaped cell is always the same distance. This helps keep distance calculations simple and offers a more rounded and fluid feeling to movement options from cell to cell compared to a square grid with orthogonal movement, as hexagon grids have 6 orthogonal movement directions compared to the square grid’s 4.

Game designer Jeremy Lennert (2018) makes a comment about some other strengths of the hexagon grids on an online forum of the website BoardGameGeek, stating hex grids give a better approximation of straight-line distances and have more symmetries than the square grid. (Lennert 2018.)

In summary, the hexagon grid's nature brings efficiency in tessellation, consistent distances, and adaptability to curvature. The hexagon's six identical neighbors and defined orthogonal movement directions provide a fluid and rounded experience, making it a superior choice in various applications.

3.2.2 Weaknesses of a hexagon grid

Wargame author and computer scientist D.Ezra Sidran's (2018) website for their game General Staff criticizes the hexagon grid for causing what they call the "drunken hexagon walk" syndrome. The aim in the example (Picture 11) is to move straight from point A to point B at a thirty-degree angle, and from point A to point C at a 90-degree angle. However, the path becomes convoluted, involving twists and turns, resembling a drunk staggering across the street. This results in traversing more terrain than necessary. Despite the hex's potential for simplifying movement, it often leads to impractical routes, far from the straightforward approach of real military units. (Sidran 2018.)



PICTURE 11. How movement paths can alternate on a hexagon grid (Sidran 2018).

This is criticism that is like what has been said for the square grid; reaching a location at an angle that does not fit the grid type means it can often only be reached by this type of zig zag movement. They conclude by saying that hexagons are a "necessary evil" for board war games to simplify movement calculations, but they see no reason to use hexagons with a strategy or war video game, since a computer can easily handle calculations for precise distances for

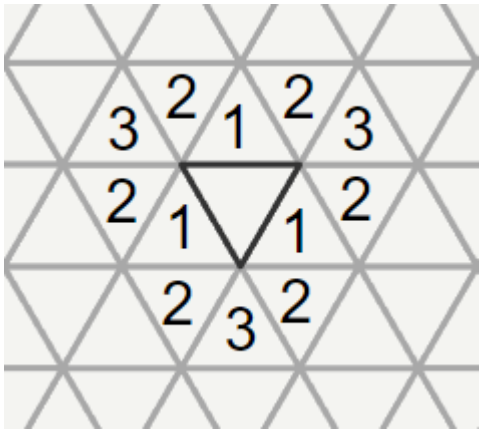
diagonal movement on a square grid or even if units move freely without a grid like in their game General Staff. (Sidran 2018.)

The research paper published by the Indian Institute of Technology (Apte et al. 2013) discusses the possibility of the hexagon grid being the most accurate and efficient grid type and raises some points for why the hexagon grid is not more popular despite their research indicating it as superior to other grid types. They attribute this to a lack of infrastructure as well as commercial availability of software and display systems that support hexagons. “The square grid is the grid that has been prevalently used since ages hence all supporting infrastructure and display systems follow the square grid.” (Apte et al. 2013, 27.)

3.3 Triangle grids

The equilateral triangle is a shape that is far below the shapes mentioned so far in terms of popularity for a grid shape. It is less common in videogames in comparison to the others, as well as in other areas besides 3d mapping or graphics (Patel 2006). The triangle also has the least direct neighbors, leading to less gameplay options than the square or hexagon grids.

A user by the name of Rogryg (2020) comments about the use of triangles in videogames on a discussion thread in Reddit, and how it has flaws seen in both the square grids and hexagon grids. Like the hexagon grid, it does not fit well to Cartesian coordinates, in this case meaning it does not tile as easily along the x and y axes. And like the square grid, the triangle grid also has neighbors on its sides and vertices, but unlike the square which can have 2 separate distances to its neighbors, the triangle grid can have 3 (Picture 12) different distances. (Rogryg 2020.)



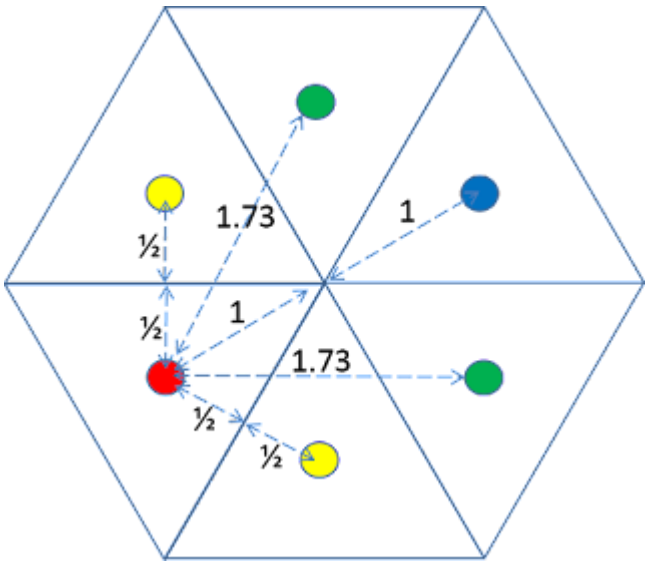
PICTURE 12. All 3 types of neighbors of a single triangle cell (Boris 2021a).

A triangle cell has a total of 12 neighbors, grouped into 3 different groups according to their distance. This makes distance calculations even more complicated with 3 different distances to account for when calculating. An option would be to only allow for orthogonal movement like the square grid, yet orthogonal movement would only have 3 movement directions. Having different distances to many kinds of neighbors is a reason why triangle grids are criticized.

Phil Dutré's (2018) blog post on Wargaming Mechanics discusses this distance counting dilemma (Picture 13).

> Taking the red dot as starting point, we see that we can get to edge-to-edge triangles (yellow dots) using distance 1. The triangle directly opposite (blue dot) requires distance 2. Both triangles that touch the starting triangle, but are not directly opposite (yellow dots), are at a distance equal to the square root of 3, or 1.73. (Dutré 2018.)

Dutré (2018) concludes how based on this, a movement system on a triangle grid could use a movement value of 1 when moving to direct neighbors and rounding the distance to the secondary neighbors to 2. Though it is well explained and reasonable, such a system has not really been used in games, and even the author suggests it should be tested first. (Dutré 2018.)

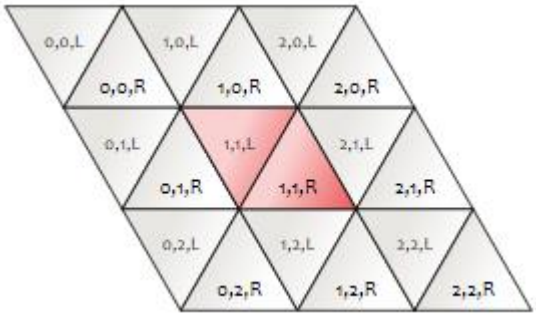


PICTURE 13. A triangle's distances to its neighbors (Dutré 2018).

Like with the hexagon grid, it is not as easy to illustrate coordinates in the triangle grid with the Cartesian coordinates. Amit Patel (2021) introduces their method of portraying coordinates on a triangle grid (Picture 14), though there are many other possible ways of doing a coordinate system for a triangle grid.

> To define triangle grid coordinates we can start with square grid coordinates, shear the squares, then split the rhombuses into two triangles. This means each square face coordinate needs to be two triangle face coordinates. I chose to label them L and R. (Patel 2021.)

This method (Picture 14) for converting a square grid into a triangle grid shown by Amit Patel (2021) shows how it can be achieved by shifting the rows or columns of the square grid and splitting each square into two triangles.



PICTURE 14. A way of defining coordinates and a triangle-based grid from a square-based grid (Patel 2021).

Though this is a good way to visualize the creation of a triangle grid and create it, as a game mechanic having to split or merge the grid cells would mean that the data contained in them would clash and raises the additional question of what should be done to circumvent this. For example, if a chess board has a piece on a square cell and that cell would be converted into two triangle cells, which triangle cell would the chess piece end up on? Or if two pieces were on two triangle cells that together originally formed a square cell and they were converted into a square cell, what would happen to the pieces on those cells?

The situation is similar with converting between triangle grid and the hexagon grid, but instead of splitting a square into two triangles, a hexagon can be split into six triangles, making the formerly mentioned issue with the data contained in the cells even more relevant. For this reason, the triangle is excluded from study of the grid conversion system. The triangle and hexagon grids also have an interesting relationship, being each other's dual tiling, meaning they can easily be portrayed as each other (Dutré 2018). A hexagon can be split into six triangles, and a triangle grid where movement is only done on the vertices of the triangle is technically a hexagon grid-based movement system.

All in all, the triangle grid is a rare sight as a primary grid type in games for grid-based movement. It shares many of the weaknesses that square and hexagon grids have and it has complications when converting it, so it is a more undesirable target for the research of the conversion system.

4 CODE DEVELOPMENT OF GRIDS

This chapter discusses the creation of square and hex grids in the Unity game engine and compares the differences in their creation processes. It will also handle the possible uses of a grid as well as compare the grids with a more mathematical viewpoint.

4.1 Creation of a grid

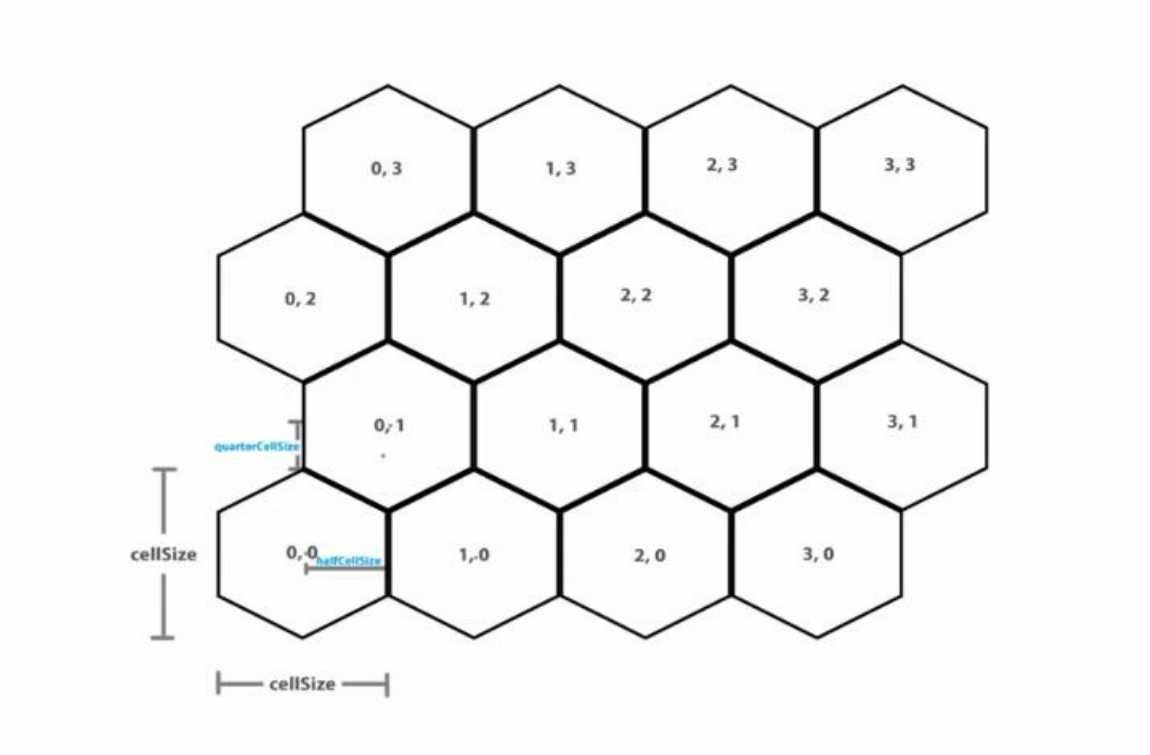
A grid can be done with many different methods without only one clearly superior answer, so for this section the creation of a square grid is showcased using a tutorial by YouTuber Code Monkey (2019a) as an example.

In code, a grid is typically a two-dimensional array of values or objects. These array items are instantiated into the playing field according to the order of the array by looping through it and distancing each object a set amount from each other. The instantiated objects can be given a mesh or a sprite for visualizing it on the screen. (Code Monkey 2019a.)

With square grids, the creation process is somewhat simple. You can create the grid object, create a two-dimensional array of them of a given size, or height and length, and then spread them to the scene to their correct positions. These positions are determined by the size of an individual cell, the total count of cells as well as the position of the specific cell in its array. This position for each cell is its position on the grid, multiplied by cell size. Cell size represents the diameter of a single cell and is used for managing the total size of the grid and the cells. (Code Monkey 2019a.)

The process is somewhat similar for the creation of a hexagon-based grid. However, the common difference in the creation process is the placement and structure of the individual cells (Picture 15). Due to the structure of a hexagon, the squares in the grid cannot simply be changed into hexagons, due to either leaving empty space between cells, or overlapping with other cells. To prevent this, the

hexagon grid can be done with each cell of either every other row or column, depending on the direction of the hexagon, being offset from each other by a set distance. This offset distance is 50% of the size of a single to the direction of said row or column. (Code Monkey 2022a.)

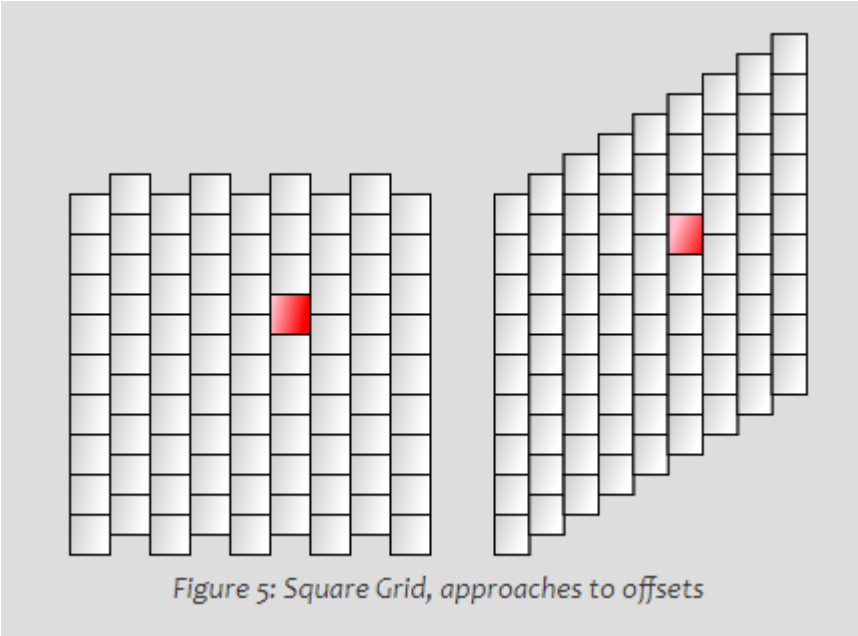


PICTURE 15. The overall offsetting to create a hexagon grid, portraying the half and quarter cell sizes (Code Monkey 2022a.)

For example, if the hexagon grid is done so that the hexagons are tiled horizontally (Picture 15), every other horizontal row would be offset by 50% of the cell size into the chosen horizontal direction, left or right. After this offset, the vertical distance between two cells' centroids on the same column should be only 75% off the cell size. The second row from the bottom (Picture 15) has its cells moved to the right by 50% of the cell size, leaving its cells seemingly between the other rows cells.

There is another way to offset a hexagon grid than offsetting every other row or column, and that is to offset every row or column by half the cell size in relation to the previous row or column like the right grid in the example (Picture 16). The method of creating a hexagon grid this way is more uniform since every row or column is offset with the same procedure, but it creates a playing field which is

less rectangular. The cells are still square shaped (Picture 16), but cells have up to 6 neighbors, and these different methods of offsetting a square grid create enough space for the cells to be changed to hexagons. This less rectangular grid will be referred to as the tilted hexagon grid.



PICTURE 16. The different approaches of offsetting a square grid to become a hexagonal grid (Patel 2006).

In code, the offsetting can be done when establishing the hexagon grid by looping through the two-dimensional array. Say if a for-loop is used, the cells can be offset by tracking if the row or column of the cell that is being established is odd or even. This can be tracked by calculating the remainder of the rows or columns index when divided by two. If the remainder is zero, the currently tracked index is even, otherwise the index is odd.

With a grid using a two-dimensional array, the first dimension of the array is typically the horizontal x-axis of the array, and the second dimension is the vertical y-axis. The two dimensions of the array can be looped through by establishing a for-loop with another for-loop inside of it. The outer loop goes through the y-axis, or the second dimension, of the grid and the inner loop goes through the x-axis, or the first dimension, of the grid with the innermost loop instantiating the grid object at the current array index. The position of the grid object on the x-axis is the index of the arrays x-axis multiplied by the cell size, and on the y-axis the

index of the arrays y-axis multiplied by the cell size. With the hexagonal grid, the cells that need to be offset will have 50% of the cell size added to their positioning at this point. The position of the whole grid can also have an additional offset, so the grid's first element isn't on the playing field's origin point, if this offset is applied for each of the grid's cells.

4.2 Use of grids and pathfinding implementation

Once the grid is set up it is possible to implement various functionalities to benefit from it, such as pathfinding and efficiently reading the player's input. Pathfinding is essential for games where units move along the grid, and reading the player's input is key to being able to interact with the grid and any objects on it.

For implementing any of these features, the grid class also needs efficient functions for returning a three-dimensional vector for the world position based on a cell's grid array position's x and y integers, as well as a method for returning x and y integers based on a three-dimensional vector position. These methods will be necessary for most functions of the grid and pathfinding, as the world position and the x and y array values of cells play a crucial role in nearly every method for identifying the cells and getting them into their correct positions. These methods and the other functions of the grid class and pathfinding class were mostly done according to Code Monkey's various tutorials.

4.2.1 Pathfinding

Pathfinding allows the game to calculate distances and optimal routes to different locations on the grid, which is essential with grid-based movement, especially if the game in question has moving CPU characters.

The method for pathfinding used in this study is the A* (A star) method. It is a method that reads the neighbors of individual cells to find the shortest and cheapest possible path to the target location. Other methods such as Breadth First Search and Dijkstra's Algorithm can also be used, but A* is best optimized for searching a route to a single location, making it a popular choice in video games for pathfinding. (Patel 2014.)

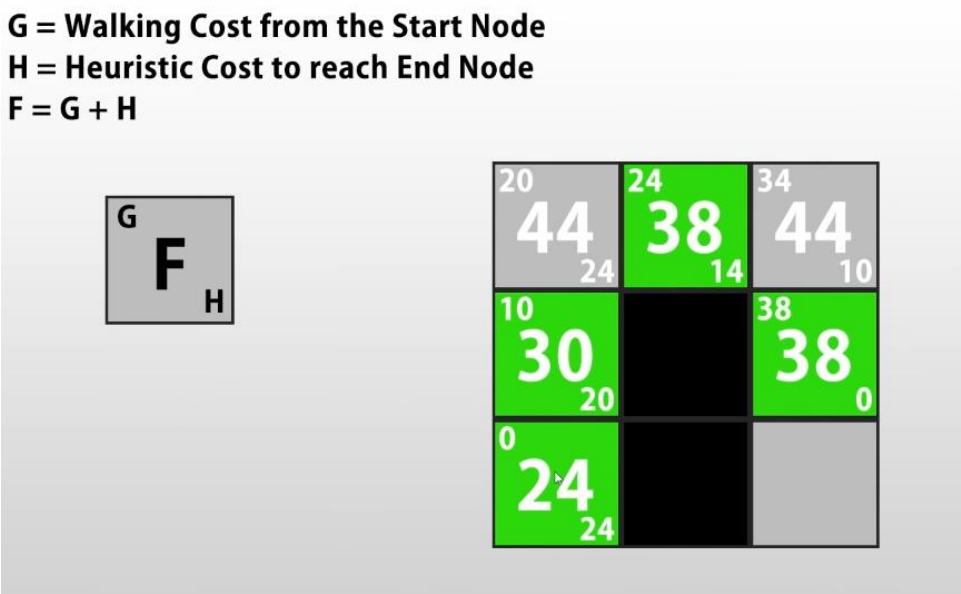
The A* method picks a cell at each step according to which has the lowest total F-value, which is a sum of two other values, G and H.

The G-value is the movement cost to move from the starting point to a given cell on the grid along the path generated to get there.

The H-value is an estimate of the total movement cost from a given cell to its destination without accounting for any form of walls or other obstacles on the path and is only a smart estimate, not necessarily representing the true cost to the target cell.

At each step the A* method picks the neighboring cell with the lowest F-value and processes the cell. It processes cells one by one until it finds the most affordable path to the target location. (GeeksForGeeks 2022.)

A square grid that allows diagonal movement can have a different movement cost for moving to a diagonal neighbor. In this example (Picture 17) by Code Monkey (2019b), the movement cost to a direct neighbor is 1 and 1.4 to a secondary neighbor and costs have been multiplied by 10 to have all the numbers be integers. In the example (Picture 17) the grey square on the left shows how each cells F, G and H-values are portrayed on the grid, the bottom left corner of the grid with the G cost of 0 is the starting cell, the green cells highlight the optimal path to the target cell and the completely dark cells are impassable obstacles.

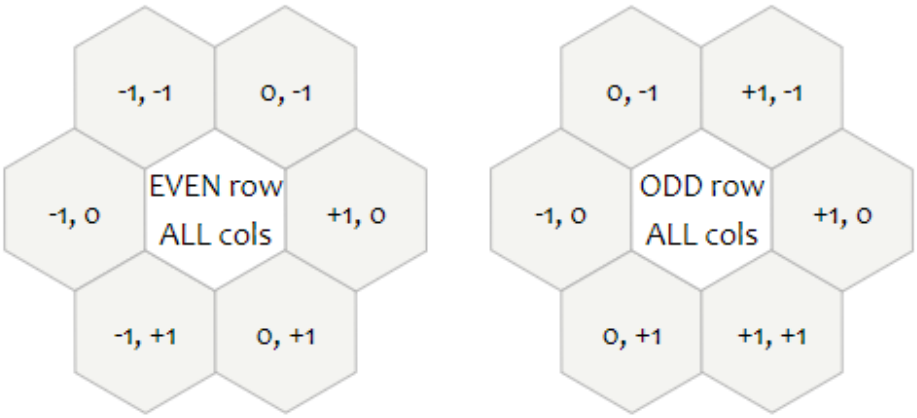


PICTURE 17. An example of A* pathfinding on a square grid with diagonal movement allowed (Code monkey 2019b).

In the example (Picture 17) it is possible to see how different the cells' values are. The starting cell has a G value 0 because no steps have been taken yet, and the target destination has a H value of 0, because the distance from that cell to the destination is 0.

Pathfinding algorithms work quite similarly on square and hexagon grids, with some minor differences. The different grids having different amounts of neighbors means the exact same method for looking for neighbors cannot be used. The square grid can look for its neighbors simply by getting the cells on both of its sides, above and below it, as well as the diagonals if wanted. These cells can be obtained by deducting or adding 1 on the current cell's array index on the x and y axes. The hexagon grid's neighbor-searching works similarly, but must account for 6 neighbors, and whether the current cell is on an odd or even row due to how the columns or rows of the grid go in an alternating pattern. (CodeMonkey 2022b.)

But since there are 4 ways of constructing this type of hexagon grid, it is hard, and likely not possible, to create a formula for getting the neighbors that would always apply for all forms of the grid (Picture 18) (Picture 19).



PICTURE 18. The sets of neighbors for a horizontally tiled grid, left cell is on an even row and the right one is on an offset odd row (Patel 2021).

PICTURE 19. The sets of neighbors for a horizontally tiled grid, left cell is on an even column that has not been offset and right one is on an offset odd column (Patel 2021).

The left number in each cell (Picture 18) (Picture 19) represents the x-axis step from the cell in the middle, and the right number represents the y-axis step from the cell in the middle. It is to be noted that in the examples of the images, the Y-axis goes downwards.

A hexagon grid with offset coordinates has 2 different coordinate sets, one for even and one for odd rows, for its 6 neighbors while the square grid only has one set of coordinates. With a hexagon grid with a doubled coordinate scheme, it is possible to have a hexagon grid with a single formula for getting all its

neighbors but offset coordinates will be preferred in this research due to resembling a square grid's coordinates more. If the neighbors of a hexagon cell can be obtained correctly, A* pathfinding works the same on the hexagon grid as on a square grid.

4.2.2 Reading input

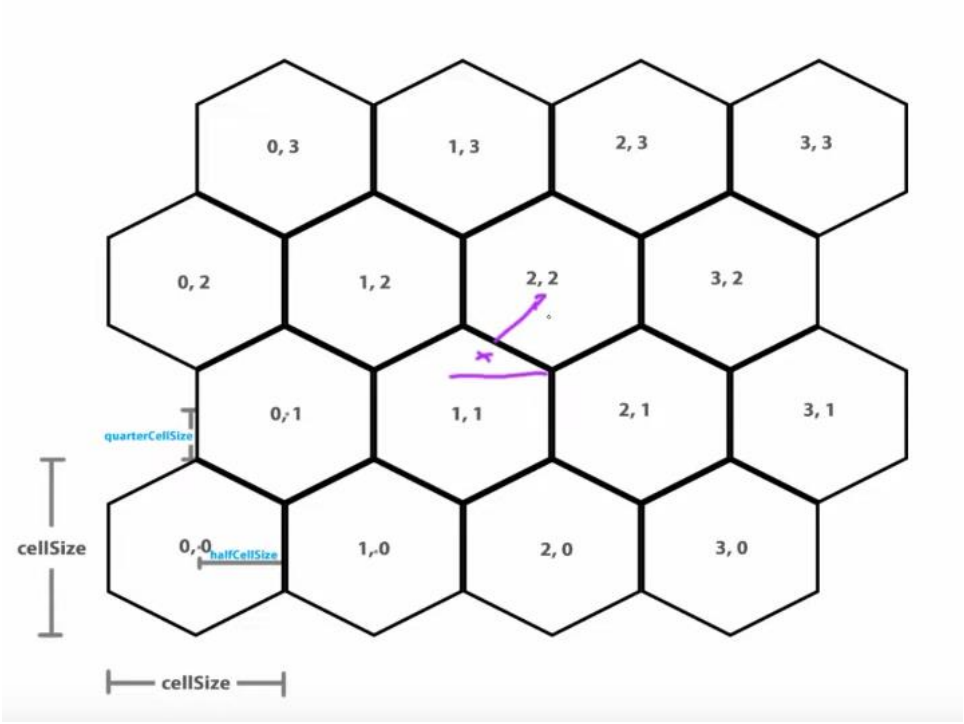
Reading the players input on the grid is essential for interacting with the grid. This can be done by reading the world coordinates of the player's input and calculating which grid cell it is on depending on the location of the grid in the scene. If the grid does not start from the playing field's origin point of position, the offset of the grid has to be accounted for when calculating the correct cell based on the input. For this reason, it's important to control where the grid construction is started and save the location. (CodeMonkey 2019a.)

On the square grid, input reading is easier due to the tiles not being offset and them being square shaped. The current cell can be determined by getting what world position the input lands on by using a method to receive the player's mouse etc. position, dividing that position's coordinates by the used cell size and rounding that to the nearest integer number. The division by the cell size is done to adjust the input's value to the grid and rounding that value to integers gives the exact cell the input landed on. (Code Monkey 2019a.)

For an example on a square grid, if the player's input lands on world position which's location is [186, 52.7] and if each cell were to have a size of 10, after division we get the values of [18.6, 5.27]. If the grid starts from the origin position of the world without any offsets, we can round those numbers to see that the input is on the cell that's position on the grid is [18, 5].

Getting the correct cell that the input lands on is not quite as simple on a hexagon grid due to the more complex borders and structure of a hexagon, as well as every other row or column being offset. This is due to how the input is typically rounded down to the nearest integer to get the position of the cell. But since the

vertices of the hexagon protrude, just rounding the input is not enough (Picture 20).



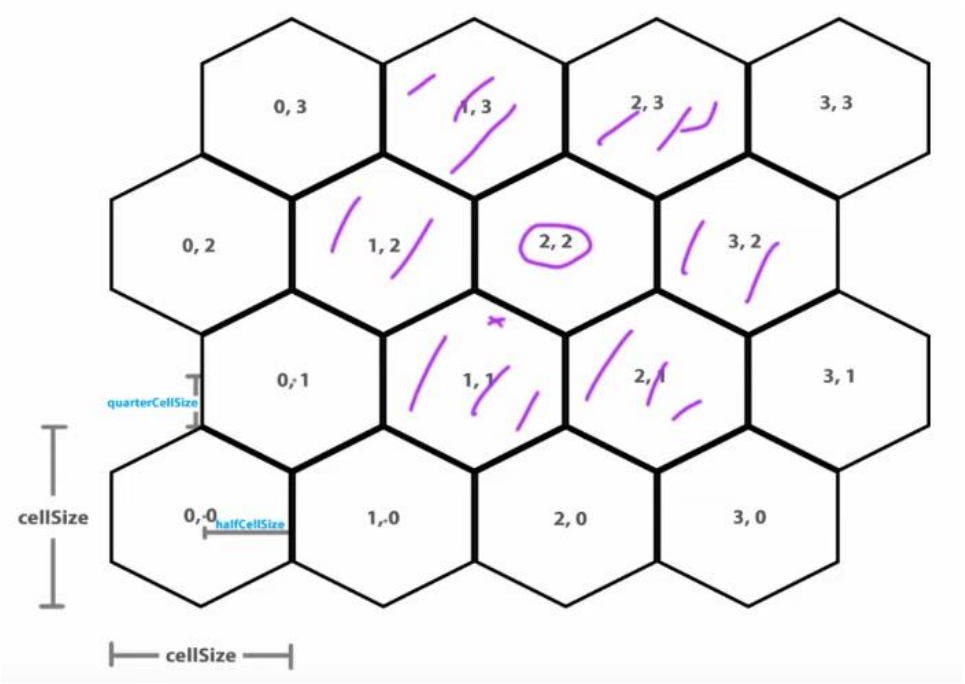
PICTURE 20. Just rounding is not enough for getting the correct cell that the input lands on with a hexagon grid (Code Monkey 2022a).

The cross (Picture 20) shows the player's input, which is above the horizontal purple line which represents the y axis position of 2 in the world. This means the input has y-axis position of 2 if it is only rounded, which would instead give the cell which the arrow is pointing towards.

Rounding the input's position is still required for getting the correct cell, but additional steps are required. The correct cell can be received by first getting the player's input, then dividing it by the cell size and rounding it to integer numbers to get an approximation of which cell the input is on and, like mentioned earlier, this may be an incorrect cell. Then a list of neighbors for this likely incorrect cell must be established. This list has variations due to there being different ways of constructing a hexagon grid, so it is important to search the correct neighbors based on the grids form, like how the grid is tiled and whether the current cell is on an offset row or column. Lastly comparing the distances of the position of the cell that was received through rounding the player's input and each neighbor's position to the originally received input's position that hasn't been rounded.

Whichever neighbor is the closest to the original input's position must be the cell that the input landed on. (Code Monkey 2022a.)

The cross in cell [1, 1] (Picture 21) shows the player's input, the circled [2, 2] is the cell that is received when the input is divided by the cell size and rounded to integers, and the hashed cells around [2, 2] are that cell's neighbors. The distances between the input and each of these cells are calculated to figure out the correct cell. The distance between the cell [1, 1] and the input is the shortest, making it the correct cell.



PICTURE 21. An example of how the correct hexagon cell can be read through player's input (Code Monkey 2022a).

With functional input reading being a necessity for interaction with a grid, it is important that it is implemented both correctly and efficiently. Having an accurate method for reading the user's input for both the square grid and the hexagon grids is crucial for studying the conversion system between the grid types, as neither grid's method for input reading works appropriately with the other grid.

5 SYSTEM OF CONVERSION BETWEEN SQUARE AND HEXAGON GRIDS

This chapter will discuss the potential of having a gameplay mechanic where the player has the option to convert the square grid area of the game to a hex grid area and vice versa. This feature is intended to be used for a turn-based strategy game with grid-based movement, where the geometrical differences of the grids as well as the shifting of the play area can be used to the players advantage when they are doing tactical decisions. This advantage and difference in the grid's current form can further be contrasted by other game mechanics, rules etc.

5.1 Square grid to hexagon grid conversion and vice-versa

Square grid conversion to hexagon grid is a topic that isn't very discussed. Most times, this might not be something that could have many uses by itself, but it could have a use as a game mechanic. This method of conversion that is utilized is essentially the same method that was used earlier to establish a hexagonal grid from a square grid.

The goal is to be able to establish a square grid, to be able to convert that to a hexagon grid of a chosen form and convert between these grids and for all grids to function properly and correctly retain their methods and functionalities.

5.1.1 Theory of conversion study process

The basis of the conversion is to shift rows or columns of the square grid in the direction that the player chooses and change the grid shape from square to hexagon. The distance that the offset creates allows the cell shape to convert to a hexagon. It is worth mentioning that the visuals of the cells must be changed as well and be the correct size in relation to each other so as not to leave any gaps on the grid. All in all, the cells need to have 3 visual variations, square, vertically tiled hexagon and a horizontally tiled hexagon.

This method of conversion from a square grid has 4 variations, differentiating between the tiling directions of vertical and horizontal as well as the two directions to offset the hexagon cells. Either the squares are converted to hexagons that have a vertex pointing up, or one that has a side pointing up. The direction of the hexagon decides whether the conversion is done by shifting columns up or down, or rows left or right. These directional shifts could also have variations. This variation could be between if only every other row is offset, or is every row offset in relation to the previous one which creates a tilted hexagon grid. The addition of tiled hexagon grid variants would increase the total of different hexagon grids to 8. But these tilted hexagon methods of tiling will mostly be omitted for this study, as the alternating offset tiling methods manage to retain a more square-like shape.

The primary method would be to shift every other row or column of the play area on either the horizontal or vertical axis respectively. This method is like how a hex grid can be created from a square grid, and reversing the conversion can be done by undoing the shifting and changing the grid game object back to the correct form.

Mike Tyka (2017), a doctor in biophysics, calls the method of conversion that is studied here “naive”, due to the aspect ratio of points being changed. His method for conversion is slightly different, but the aspect ratio of points changes in both. However, this fault may be possible to circumvent with limitations as a game mechanic.

5.1.2 Implementation

The implementation of the conversion system can vary in depth depending on how many variations of the hexagon grid will be available for conversion. There are alternating offset hexagon grids and the tilted hexagon grid, and both of these grids can be offset into vertically or horizontally tiled grids, and those grids can also be offset into either the positive or negative direction of their tiling. To clarify with an example, for the vertical tiling the positive direction is up, and the negative direction is down. All in all, this can lead to a maximum of 8 different

hexagon grid variations. Though for this research, the tilted hexagon grid variations will be omitted due to being less rectangular, totaling for a maximum 4 hexagon grid variations and the square grid.

The common parts of the grids' creation processes must be determined for finding a solution for the implementation of the conversion system. This common part in the creation process of the grids is establishing a two-dimensional array of a given size, and having it illustrated or having the grid cells instantiated for each array member. The biggest difference in the creation is the position of each cell in relation to other cells, as well as the actual visual form of the grid cell. Other big differences aren't as relevant until implementing other features such as pathfinding, although pathfinding and looking for the cells' neighbors work mostly the same as before. Ultimately, the creation of the grids is nearly the same as when creating a square, except for the positioning of the cells.

Either grid can be constructed first, and at this point, converting to the other grid type only requires changing the positioning and shape of the grid cells. The same logic that is used to offset the cells when making a hexagon grid can be used to offset the square grid's cells to convert the grid into a hexagon grid, and that logic can be used in reverse to turn the grid back to a square grid. It's important to be aware of which form the grid is converted to and save the information. This could be done by establishing an Enum variable called e.g. "GridType" which portrays the current state of the hexagon grid, then creating a variable of type GridType that is used to store the current form of the grid. This value is important to properly manage the many functions for the hexagon grid, since these methods will have a different outcome for each form of the grid. This Enum should have states for every possible grid form (Picture 22). These methods include most key methods in the grid class as well as in the pathfinding class which should have the GridType accessible through some way.

```
35 references | 12 references | 5 references | 5 references | 5 references | 5 references
public enum GridType {square, vertHexUp, vertHexDown, horiHexRight, horiHexLeft}
4 references
private GridType currentGridType = GridType.square;
```

PICTURE 22. shows how the GridType Enum was established, with currentGridType being used to keep track of it (Veeti Karilainen 2023).

Converting back to a square grid can be done by looping through the grid again and positioning each grid cell like how they are positioned when making a square grid. Each square cell will be at a position in the playing field that is the cell's position in the grid array, multiplied by cell size, with any possible world origin point offsets regarded for. With a grid starting from the world's origin point, a cell at position [2, 4] in a grid with a cell size of 10 would be in world position [20, 40].

Both the functionality for hexagon and square grids can be implemented in code, and it's possible to use the right code based on what form the grid currently is. It's undesired to use e.g. pathfinding methods of a square grid with a hexagon grid, due to them not functioning as well as the functions designed for them.

Methods for pathfinding, grid construction as well as the grids own methods for both square and hexagon grids should be combined to make methods for pathfinding and grid construction that work with both grid types.

For example, looking for a cell's neighbors could be a method with a switch case, which executes correct code for the grid's current form based on the GridType Enum. This way the neighbor searching method's switch case can obtain a list of the correct neighbors to a cell regardless of the form of the grid. The GridType Enum value must be adjusted to always appropriately describe the form of the grid so the methods of the grid function properly. The Enum value can be adjusted whenever methods are called to change the form of the grid.

5.2 Conversion system's impact in-game

The differences in square and hexagon grids can lead to different gameplay situations, and the grid conversion system could be used to highlight these strengths and weaknesses in gameplay. Generally, the traits of the grids have

roused discussion over which grid is superior, though both grids do have strengths as well as weaknesses.

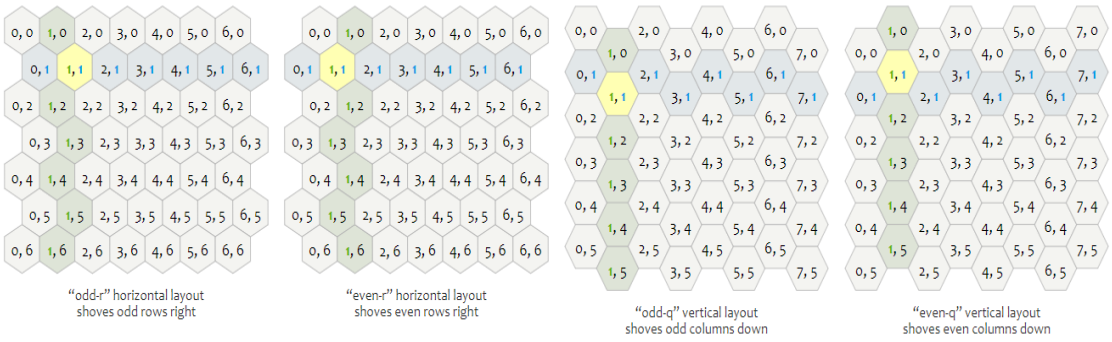
It would be unfair to say that one grid type should be completely overlooked as an option for game design simply due to another grid having a superior feature. Board game design enthusiast Danny Truong shares their preferences on their YouTube channel, Board Game Sanctuary:

> I personally gravitate towards the hexagon, just because the shape is more organic when it comes to map building but also the options that players have. But I also like games that use the square effectively, and when games can kind of marry the shape of the game to the mechanisms and their theme and they can do it well, then that says to me that it doesn't really matter what shape the game is, the game is going to be fun regardless. (Truong 2022.)

Either grid is a perfectly valid option from a game design standpoint. Both grid types have their traits and ideal use-cases, further implicating that being able to convert from a grid to a different type can have significance as a game mechanic. Game mechanics and levels could be designed with either or both grids' traits in mind, giving the player the freedom to capitalize on the conversion system as they see fit.

5.2.1 Gameplay impact of the conversion system

The conversion system can visibly affect gameplay in turn-based strategy games. The conversion changes movement options, the number of neighbors a cell has and most importantly, the cells' shape, leading to different traits of the grids that affect gameplay. The example (Picture 23) shows the four types of alternating offset hexagon grids with cell 1,1 highlighted in yellow. In each grid, the cell highlighted in yellow has a different set of direct neighbors. In a strategy game where unit positioning is important, the player can utilize this and convert the grid to give them an advantage.



PICTURE 23. Different hexagon grid variations' neighbor sets (Patel 2021).

In a square grid, the same cell [1, 1] only has 4 direct neighbors. When the grid is converted to a hexagon grid, the cell now has direct access to two cells that were previously only secondary neighbors. This fact can be especially exploitable when using strictly orthogonal movement. Cells that were previously 2 units of measurement away on a square grid could now be only 1 unit of measurement away. Cell [1, 1] has a secondary neighbor [2, 2] (Picture 24), which is a direct neighbor in the leftmost grid of the previous example (Picture 23).

0,0	1,0	2,0	3,0	4,0	5,0
0,1	1,1	2,1	3,1	4,1	5,1
0,2	1,2	2,2	3,2	4,2	5,2
0,3	1,3	2,3	3,3	4,3	5,3
0,4	1,4	2,4	3,4	4,4	5,4

PICTURE 24. Example of the square grid with coordinates (Patel 2021).

This clear change in accessible neighbors further indicates the effect of the grid conversion as a game mechanic. If the player can control the grid shape, they can also directly control the shape of the entire playing field along with everything on it. Being able to shift the positions of enemies, obstacles, objectives and other things on the playing field could be a powerful tool in the hands of the player without giving too much direct control. In army-centric turn-based strategy games the shifting can create openings through enemy formations or get

some player units in better positions to act. A unit that is surrounded from the directions of all direct neighbors on a square grid could find an opening to break through with the conversion to a hex grid.

Another example of being able to use this system to travel more directly to a location would be movement from cell [0, 0] to cell [5, 4] (Picture 24)**.** This travel takes 9 steps, but in the furthest right hexagon grid (Picture 23), this travel only takes 6 steps.

In many turn-based strategy role playing games, units with average mobility usually only have between 4-6 movement, with some terrain cells hindering mobility, so using your movement optimally can be important. For example, in Fire Emblem Engage, published by Nintendo in 2023, the protagonist starts the game in the “Dragon Child” class, which can only move 4 spaces on a square grid that does not allow diagonal movement. “Brigandine: The Legend of Runersia” is a turn-based strategy role playing game published by Happinet and played on a hexagon grid where many units, like five of the games six alternative protagonists, can also only move 4 spaces in the beginning.

These examples will heavily reference the Fire Emblem franchise developed by Intelligent Systems and published by Nintendo, as it is one of the most prolific turn-based strategy franchises, being active since the 1990 and with 17 mainline titles published, making it an excellent example.

In some if not most games of this genre, there are ways to improve units’ mobility either permanently or temporarily, but mobility is still a finite resource in these games that is important to use efficiently each turn. The Fire Emblem franchise has a movement boosting consumable item called Boots, that increases a single unit’s mobility by 1-4 points, differentiating slightly between titles. The “strongest” effect that the Boots have had was in “Fire Emblem: Shadow Dragon and The Blade of Light” released in 1990, and “Fire Emblem: Mystery of the Emblem” released in 1994, but it is possible to obtain only 2 of these items in these games, showing how rare of a resource movement boosting mechanics typically are.

To highlight the importance of movement in games of this genre, it is important to have a grasp on the typical size of the playing grids. A Reddit user by the name of TheEntireRomanArmy (2019) has calculated the average size of a level, or a map, in every Fire Emblem game that was released until 2019. All in all, the average map size across the franchise is 696.05 cells per map, with the lowest being the 7th game in the series, “Fire Emblem and the Blazing Blade” with an average map size of 398.3 cells, and the highest average being the 4th game in the series, “Fire Emblem: Genealogy of The Holy War” having an average map size of 3925.3 cells per map. Games of this genre can have maps that are hundreds or even thousands of cells large, so making the most of the very finite movement of your units is a crucial factor to playing well. Many games incentivize such optimal play with time sensitive optional objectives to reward the player, an example of such an objective could be having to clear the map objective in a set number of turns.

5.2.2 Usage of the conversion system as a game mechanic

Efficient game design is necessary to have this game mechanic shine in a video game, since like mentioned earlier, hexagon grids typically help you to get to some further locations faster than the square grid while having more neighbors available for strategic decisions. So as not to make the game balance too heavily favor the hexagon grid, multiple game mechanics should factor in the grid type in use, and some game mechanics could circumvent some of the grids’ weaknesses altogether, while bringing in something new to the genre. All this design and balancing should incentivize the player to use the right grid type when necessary and hopefully not stick to a single grid. The aim is not to make playing on a square grid identical to a hexagon grid, but to make them both be balanced while embracing the different traits of the grid.

Some things that should be accounted for and resolved with game design include: the square grid’s diagonal neighbors, distances and its less efficient movement compared to the hexagon, the hexagon grid’s movement paths alternating in some cases and the different neighbor counts on different grids. An additional problem this could fix is one that is quite prominent at least in the Fire

Emblem series, the very low movement of armored units which usually renders the class nearly useless.

Starting with the diagonals of a square grid, this trait could be accentuated by having only some units or opportunities to move diagonally for the same cost as moving to a direct neighbor, in which case movement would be cheaper on the square grid. This should be handed out very sparingly but would allow units to have more movement options, while giving players more incentive to proceed on the map with the square grid.

Overall, the hexagon grid's movement will still be more efficient, so the alternating movement of the hexagon grid could be addressed for balance. To be more precise, the alternating movement refers to the "drunken hexagon walk" that Sidran (2018) discussed. This mechanic could be referred to as momentum, where the direction and straightness of a unit's movement could affect gameplay. In some cases, movement that occurs as straight as possible could be beneficial, and in other, slightly rarer, cases the alternating movement could be beneficial. The argument against the alternating movement on a hexagon grid is the same as what Sidran (2018) used to criticize the hexagon grid. This mechanic would be more beneficial for the square grid, where the straight movement axes of the grid could be more easily benefitted from to charge forward. Since the hexagon grid has more movement options, it also has more chances for movement on the grid to become slightly alternating. Though square grid also has more than enough chances for movement paths to become alternating, this mechanic should favor the square grid slightly more to further contrast the use cases of the grids. Ultimately both grids should benefit from movement being committed as straight as possible, with the square grid benefitting from it more. These benefits could include boosts to combat performance, such as increased damage, hit chance or dodge chance.

In a typical combat forecast in this genre of games (Picture 25), both units have a calculated damage value "Dmg", hit probability "Hit" and chance of landing a critical strike "Crit" with the arrows showing the flow of attacks and the health points "HP" of the units showing their estimated condition after the combat. All these values are calculated with every stat and skill in mind that affects it, in the

example (Picture 25) the unit on the left would do 10 damage per hit, which is a sum of their strength and the attack of the used weapon minus the defensive value of the enemy.

PICTURE 25. An example of a combat forecast from Fire Emblem Engage, published by Nintendo in 2023 (Nintendo 2022).

To put it briefly, in games of this genre the aim is to take down the opposing units, and a unit can initiate combat against an opposing unit when the enemy is within attack range. If the enemy unit's attack range also reaches back, they will take turns attacking each other, with the initiator typically getting to attack first. Various stats determine the outcome of combat, such as the units' personal stats, skills, weapons and other things. Units deal damage depending on stats that affect attack, and attacks have a percent chance to hit or miss, with various stats affecting this.

This momentum mechanic could both give boosts to damage dealt, as well as make it more likely for the initiating unit to land a hit or dodge the enemy's counterattack. The momentum of movement could be measured by how many turns are done while moving, in comparison to how many cells are traversed. Direct and straight movement should more often grant bigger bonuses to highlight the square grid, but some skills could make units benefit from taking many turns instead, just to divert from the typical use of the mechanic. This could divide units into categories, with some units preferring alternating and some direct movement. This would further make specific units stand out more on the hex grid and

others more on the square grid, placing importance on the player's tactical choice of grid.

The next major difference is the different numbers of neighbors on different grids. Besides movement options, this also affects actions that target multiple cells at a time, like cells within a certain range or cells within the secondary or primary neighborhood. Another mechanic to balance the neighbor counts could be skills that directly get affected by the neighbor count, or number of neighboring ally or enemy units.

Skills in this genre of games are typically effects that can be specific to a character, learned by leveling up or gained by some other method such as using an item. These effects usually either provide a stable passive effect, such as improving stats or giving additional experience points, or they possess a condition which activates an effect. An example of a conditional skill could be the "Rallying Cry" from the Fire Emblem Fates games released in 2015 and published by Nintendo, which makes allied units within a range of 2 deal 2 more damage when they damage an opponent, and an example of a passive skill could be the "HP+5" skill present in the same game, which permanently grants 5 health points to the unit with the skill. The introduction of skills that factor in or rely on the shape of the grid or the number of neighbors could incentivize strategies with a certain grid. An example of such skill could be a conditional skill which boosts the unit's damage dealt for each neighboring tile with an enemy in it. This skill's potential would directly be tied to the current form of the grid, and such a skill could perform better on a hexagon grid due to a hexagon cell having up to 6 neighbors. These skills could vary in their effects and form a range of skills, some of which incentivize the use of the square grid, and some which incentivize the use of the hexagon grid.

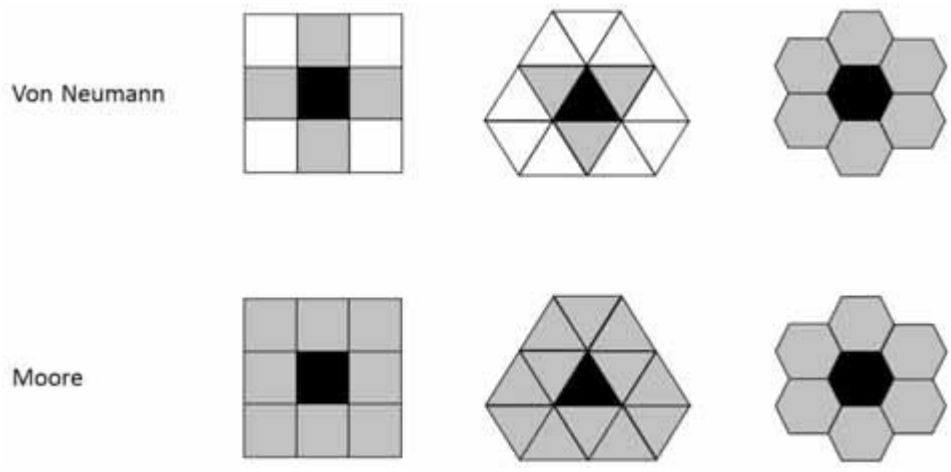
Many games of this genre feature both ranged actions, meaning actions that reach further than to a direct neighbor, as well as actions known as "AoE" actions, that target a wider area around a targeted cell or the user. Distance calculations for ranged attacks typically work the same as for movement calculations. AoE actions, standing for "area of effect" actions, target multiple cells in a

shape, such as a line of cells or every cell within a certain range of the targeted cell.

In the example (Picture 26), the area highlighted in red is the effective range of the action centered around the unit using it. If the action reaches a specific cell within that range, it is categorized as a ranged action. If it reaches multiple or all cells within that range, it would be categorized as an AoE action.

PICTURE 26. An example of ranged actions from the “Brigandine: Legends of Runersia” video game released in 2020 (Peter Sahui 2021).

Ranged attacks are typically more efficient on a hexagon grid due to the same reason as movement is, it is possible to reach further and have more potential targets. On a square grid, a typical ranged attack with 2 range can reach a total of 12 cells around it, whereas a similar ranged attack on a hexagon grid reaches 18 cells around it. This is a great asset for the hex grid, but ranged or AoE actions could be made more effective for the square grid. These actions could be split into effects with a range, and effects with a radius. Range effects would typically be more effective when using a hexagon grid and would allow actions to reach a set number of cells away from the user. Radius effects would reach a “neighborhood” around the target cell, this neighborhood being the Moore neighborhood, or every cell around the target cell (Picture 27).



PICTURE 27. Different neighborhoods for different grids, with the darkened cells being the middlemost cell's neighbors (Daniel Lückerath 2020).

Some actions targeting strictly the Moore neighborhood would make them more efficient to use on a square grid, due to these actions reaching all 8 of the square cell's neighbors, and all 6 of the hexagon cell's neighbors, with the square being able to reach more cells.

Other balancing game mechanics could be implemented too, these are just concepts to emphasize the differences between the grids while making both grids have distinct strengths. All in all, these mechanics make the grids stand out, with the hexagon grid remaining more consistent and the square grid potentially having more distinct highs and lows in efficiency.

The moment of activating the conversion is something that should also be designed properly. Things such as the frequency of conversion occasions, potential limits of its uses, and the flow of activating it. These are mostly case specific details, but most importantly the activation should offer a preview to see how the conversion would affect the playing field, especially if the number of times it can be used is limited. Conversion occasions could be present during the enemy's turn as well in turn-based strategy games, but it has the potential to cause issues due to the enemies having to reconfigure their path.

6 CONCLUSIONS AND DISCUSSION

A 2D game prototype was successfully created using the Unity game engine to study the creation process and the potential of the conversion system. This game prototype is a turn-based strategy game where movement happens on a grid. The conversion system was implemented successfully, with the functionality of square and hexagon grids being easily accessible to the correct grid types. The grid can be freely changed between grid types for the sake of demonstration, but an actual game would likely restrict this more. Additional game mechanics were not implemented for the sake of balancing the conversion system in the prototype, as the aim was to at least get a shallow grasp on the system's effects on gameplay, and more importantly to better understand and innovate the creation process. That is why the conversion system in the prototype does not quite reach its full potential, further highlighting the differences between the grids and at times making the comparative efficiency of the grids unbalanced. In the prototype the player controls a small squad of units, moving them on the grid with the aim of defeating the computer-controlled opposing squad.

Through personal playtesting some main takeaways were gathered about the conversion system, both positive and critical.

The conversion system was especially effective when moving units with extremely low movement, as efficient use could increase the distance traveled. The prototype's mage class could only move a single cell at a time, so switching to a hexagon grid could allow them to move to a cell that was previously out of reach, effectively doubling the distance traveled from a single cell to two.

The system proved quite useful when making a defensive formation of units. The difference in the number of neighbors leads to different strategies being available, leaving freedom for the player to choose which grids defensive traits they prefer. A square grid could form a more solid and safe wall of units, and the hexagon grid had a higher risk but freer playstyle.

Another important thing that was noticed is the lack of indication for the conversion system being available. Though this was just a prototype, even then it was noticeable that some indication, response and limitation for using the system would greatly improve the experience of using the conversion system.

The methods of hexagon and square grids being combined and working based on the current grid form helped make the code less messy, and the execution of code was more streamlined and easier to understand. These methods were modified based on the methods of Code Monkey's educative video tutorials about the square grid, hexagon grid, pathfinding and pathfinding on a hexagon grid.

As effective and functional as the conversion system turned out, many questions remain regarding it, including how the system should behave with buildings and walls in the levels, as well as many potential tweaks and additions that could be made to it as a game mechanic. A part of the reason for this is that not much research or discussion of a similar system, let alone as a game mechanic, was found. As for the reason why that is, there are many possibilities, but it is difficult to pinpoint any exact reasons. It could be that even if a similar mechanic had been developed, it would still have been better to simply scrap the mechanic and stick to using a single type of grid.

Many elements were omitted from the system due to the scale of the project as well as personal judgement. Elements such as having the conversion work with triangles as well, or the other forms of the hexagon grid could work seamlessly since the conversion system implementation was successful.

And although the system was implemented successfully, its effectiveness as a game mechanic in a full-fledged game project can't be said for certain. The time frame of the completed prototype was limited, and most of the balancing game mechanics did not make it into the prototype. On top of that it was not playtested as properly as it could have been. With the balancing game mechanics properly implemented, the system could be much better to use with more proper incentive to use different grid types. At the very least, this conversion system could potentially prove useful as an option for games, allowing players to play a

game on their preferred grid type by adjusting the settings. With a larger scope as a part of a bigger project there is potential to develop this system further into a proper, unique game mechanic.

REFERENCES

Apte, M., Agarwadkar, Y. Y.Y., Azmi, S., Inamdar, A. B. 2013. Second International Conference on Recent Trends in Information Technology and Computer Science (ICRTITCS-2012). Mumbai, India

Baugh, L.S. 2022. Hexagon. Britannica. 17.10.2022. Read on 15.9.2023 https://www.britannica.com/science/hexagon/additional-info#history

Bond, J.J. n.d. Orthogonal. Board and Pieces. Read on 18.11.2023 https://sites.google.com/site/boardandpieces/terminology/orthogonal

Boris. 2021a. Some Triangle Grid Extensions. BorisTheBrave. 27.5.2021. Read on 15.11.2023 https://www.boristhebrave.com/2021/05/27/some-triangle-grid-extensions/

Boris. 2021b. Triangle Grids. BorisTheBrave. 23.5.2021. Read on 24.9.2023 https://www.boristhebrave.com/2021/05/23/triangle-grids/

Code Monkey. 2019a. Grid System in Unity (Heatmap, Pathfinding, Building Area). YouTube. 4.10.2019. Watched on 5.11.2023 https://youtu.be/waEsGu--9P8?si=NzRZVosP9MIWHmmp

Code Monkey. 2019b. A* Pathfinding in Unity. YouTube. 20.10.2019. Watched on 10.11.2023 https://youtu.be/alU04hvz6L4?si=QbdipCzvN3RlEmr-

Code Monkey. 2022a. How to make a Hex Grid System Unity Tutorial! (Like Civilization, Opus Magnum, Gloomhaven). YouTube. 3.10.2022. Watched on 14.11.2023 https://youtu.be/0nXmuJuWS8I?si=DxWN1kA-3xIp6ey7

Code Monkey. 2022b. Unity Pathfinding on a Hex Grid System! YouTube. 5.10.2022. Watched on 17.11.2023 https://youtu.be/XqEBu3un1ik?si=n9jxErozouA0BHXp

Dutré, P. 2018. Triangular Grids. Wargaming Mechanics. 7.8.2018. Read on 6.2023
https://wargaming-mechanics.blogspot.com/2018/08/triangular-grids.html

GeeksForGeeks. 2022. A* Search Algorithm. Geeks for Geeks. 13.6.2022. Read on 5.12.2023
https://www.geeksforgeeks.org/a-search-algorithm/

Graham. 2010. An Introduction to Hexagonal Geometry. Hexnet 16.4.2010. Read on 23.9.2023
https://hexnet.org/content/hexagonal-geometry

Hill, R. 2016. Hex vs Square. Horrible Pain. 6.3.2016. Read on 15.10.2023
http://horriblepain.com/2016/03/hex-vs-square/

Lennert, J. 2018. Hex or square? BoardGameGeek. 11.10.2018. Read on 24.9.2023
https://boardgamegeek.com/thread/2074053/hex-or-square

Patel, A. 2006. Amit's Thoughts on Grids. Red Blob Games. 9.1.2006. Read on 23.9.2023
http://www-cs-students.stanford.edu/~amitp/game-programming/grids/

Patel, A. 2013. Hexagonal Grids. Red Blob Games. Read on 23.9.2023
https://www.redblobgames.com/grids/hexagons/

Patel, A. 2014. Introduction to the A* Algorithm. Red Blob Games. 26.5.2014. Read on 28.11.2014
https://www.redblobgames.com/pathfinding/a-star/introduction.html

Patel, A. 2021. Grid parts and relationships. Red Blob Games. Read on 23.9.2023
https://www.redblobgames.com/grids/parts/

Pierce, R. 2023, Tessellation. 18.6.2023. Read on 22.9.2023
https://www.mathsisfun.com/geometry/tessellation.html

Rogryg. 2020. Triangular Grids - there are almost no games that use it. Only hex and square grids. Have you heard of any games using triangles, do you have experience of playing/developing these? Reddit. 10.6.2020. Read on 27.9.2023
https://www.reddit.com/r/gamedev/comments/hokdvq/triangular_grids_there_are_almost_no_games_that/

Sidran, E. 2018. The Problem With Hexagons. General Staff. 14.12.2018. Read on 10.10.2023
https://www.general-staff.com/the-problem-with-hexagons/

Strategywiki. N.d. Nobunaga's Ambition. Read on 17.12.2023
https://strategywiki.org/wiki/Nobunaga%27s_Ambition

Strimas-mackey, M. 2020, Fishnets and Honeycomb: Square vs. Hexagonal Spatial Grids. Strimas. 20.3.2020. Read on 24.9.2023
https://strimas.com/post/hexagonal-grids/

TheEntireRomanArmy. 2019. I manually figured out the average map size of every Fire Emblem game. Reddit. 1.7.2019. Read on 11.12.2023
https://www.reddit.com/r/fireemblem/comments/c7mzny/i_manually_figured_out_the_average_map_size_of/

Tobin, J.C. 2020. 6 Hex Empire Games on Our Radar. CryptoCrusades. 26.5.2020. Read on 15.1.2024
https://cryptocrusades.io/blog/6-hex-empire-games-on-our-radar/

Truong, D. 2022. Are Squares and Hexagons BAD for Board Gaming? YouTube. 21.5.2022. Watched on 2.10.2023
https://youtu.be/YF9_Tk3je6c?si=pxcDKVwoL6dpum4d

Tulleken, H. 2014. 20 Fun Grid Facts (Hex Grids). Game Developer. 12.9.2014. Read on 15.9.2023
https://www.gamedeveloper.com/design/20-fun-grid-facts-hex-grids-#close-modal

Tyka, M. 2017. Square to Hex. Mike Tyka. 11.3.2017. Read on 1.10.2023
https://mtyka.github.io/graphics/2017/03/11/square-to-hex.html

Wolf, M.J.P. 2012. Before the crash: early video game history. Michigan: Wayne State University Press