# Junk Art

## A digital board game adaptation

## Draft final report

Thomas Nagel

February 2024

Content

# 0 Page/word limits

The rubric for the final report lists page limits for each report section, rather than the word limits introduced for the preliminary report. In order to be able to present relevant figures within the text, rather than in a separate appendix, this submission will assume the word count limits from the preliminary report are still valid, and extrapolate those limits to the sections not included in that previous submission, viz.:

- Introduction: 286 words (2 pages/1000 words limit)
- Literature review: 1950 words (5 pages/2500 words limit)
- Design: 956 words (4 pages/2000 words limit)
- Implementation: 1422 words (3 pages/1500 words limit)
- Evaluation: *incomplete* words (3 pages/1500 words limit)
- Conclusion: *incomplete* words (2 pages/1000 words limit)

# 1 Introduction

This project is based on the 'physics-based game' template, which requires a short, but compelling game using physics modelling as a key element of gameplay.

My proposed design will be based on simulating a tabletop board game. Board gaming is an area that holds especial interest for me, as a structure around which to base social engagements, and for the mental – and occasionally physical – challenges such games provide.

While board games are inherently physical items, a subset – known as dexterity games – make explicit use of their physicality. Such games may challenge players to manoeuvre components using manual dexterity with varying objectives, such as building towers, accurately flicking pieces around a game space, or carefully manoeuvring components using tools. Well-known examples of games in this genre include *Jenga*, *Operation*, and *Subbuteo*. This focus on the physical interaction of game components may make a dexterity board game a suitable influence for a project conforming to the provided template of a 'physics-based computer game'.

The project template specifies that the end product be easy to pick up and play. It is therefore assumed that the users (players) will not necessarily be 'hardcore' gamers, familiar with the conventions and control schemes of modern PC gaming. The users may, or may not, have familiarity with the type of physical game upon which the software will be based. Both the gameplay, and the methods of control and interaction must therefore be reasonably intuitive, or quickly learnable. The users may not have access to advanced computer hardware, and may not have use of control pads or other gaming-specific input methods. It will be important that the software can run on a reasonably specified machine, and can be used with mouse-and-keyboard inputs.

# 2 Literature review

## 2.1 Board games through history

People have been playing board games for thousands of years and throughout the world. Archaeological examples exist from ancient cultures, and records show the evolution and propagation of games throughout history. Examples of such ancient games include *Go* (also known as *Baduk*) originating around the 4th millennium BCE in China [1], the *Royal Game of Ur* from c.2600 BCE in Mesopotamia [2] and *Senet* from a similar era in Egypt [3].

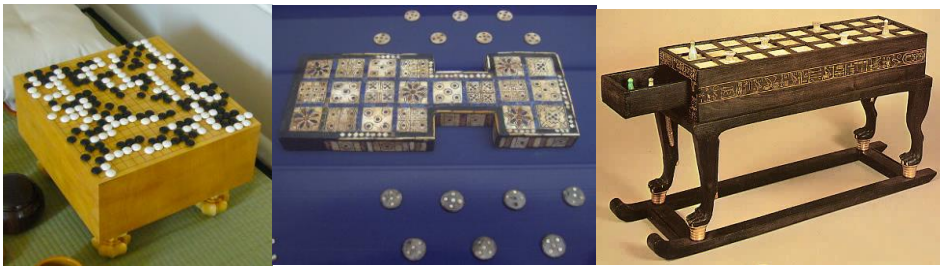

*Figure 1: [L-R] Go, Royal Game of Ur, and Senet board games*

The industrial revolution and the attendant advent of commercial mass production led to what is considered the first 'golden age' of board games in the late 19th and early 20th centuries [4]. The increased availability of games included affordable versions of older games, such as the medieval *Game of the Goose* – first recorded in 1480 [5], as well as new creations such as Elizabeth Magie's *The Landlord's Game* in 1904, now better known by its popular descendant *Monopoly* [6].


*Figure 2: [L] The Landlord's Game (1904), and [R] a 1910 publication of the Game of the Goose*

Since the 1990s the so-called 'board game renaissance', or 'second golden age' has seen a resurgence in the popularity and innovation of modern tabletop gaming, originating primarily in Germany, but now spread around the world [7]. This renaissance has coincided with the decline of certain types of social venues, causing a rise in a new type of 'third space' dedicated – or at least welcoming – to tabletop gaming [8].

## 2.2 Digital board games

There is little in the academic literature relevant to designing a project such as this; instead, examples of existing similar projects will be evaluated.

The resurgence of interest in board games in recent decades, noted above, has coincided with the expanding availability and capabilities of home computing devices, and of the internet. This has led to several instances of digital adaptations of extant board games, as well as generic platforms for hosting various games through a single interface.

*Tabletop Simulator* [9] is a popular example of the latter. This software is a physics sandbox, designed as a platform for playing board games online. The software by default only includes a handful of public-domain games such as chess, but includes the option for users to import assets for other games. With a few exceptions, game rules and mechanics beyond simple physics are not implemented; the software merely provides a world in which players can manipulate components as on a real table. This makes for a very versatile platform in which a variety of games can be played, however it requires at least one – and ideally all – of the players to be familiar with the game and implement all game rules manually.

*Tabletop Simulator* implements a functional 3D environment and physics engine, in which components are rendered and are subject to both intrinsic forces such as gravity and friction, and extrinsic forces from player interaction. Game pieces will collide with each other, and elastic collisions are modelled convincingly. Camera controls are intuitive to players familiar with using mouse-and-keyboard in other 3D computer games. When it comes to manipulating components within the game world, however, the complexity of the control

scheme increases. Due to the software's generic nature, there are many options for moving, rotating, flipping, and viewing components, and these are not well communicated by the user interface. The software's external knowledge base contains several pages on the player controls [10].

As noted, the physics engine within the software is functional, however it does not accommodate certain dexterity-type games well. Flicking games, such as *Crokinole* function reasonably; the interface provides a method for the player to impart a flicking force parallel to the ground, and objects will slide, roll, and collide realistically. Stacking games, such as *Jenga*, are virtually impossible to play using this software, due to the difficulty of manoeuvring held objects with six degrees of freedom. Objects can only be rotated in discrete intervals – the rotation angle can be set in the interface, but only within a small number of options. Movement of objects on the vertical axis is very restricted – a lift height can be set in the interface, and objects are pulled to this heigh while held and subjected to gravity when released. This does not allow for careful placing of objects atop each other.


*Figure 3: Tabletop Simulator, showing the 3D environment and force input interface for a flicking game*

In contrast to *Tabletop Simulator*, which is a piece of software sold to users and running on the users' machines, *Board Game Arena* [11] is an online service, supported by a subscription model. This service allows users to play board games with others through a web browser, with a predefined, though large, selection of board games implemented. Each game implementation is developed to enforce game rules, and automate game mechanics wherever possible. The engine used to implement each game is two-dimensional, with no physics simulation. This is thus an unsuitable platform for implementing adaptations of any dexterity games. The fact that the service hosts a large number of games is beneficial to its subscribers, but it does mean that the interface is somewhat generic, and it can take some time for a user to become familiar with how a particular game is implemented, even if they are familiar with the rules of the game itself.


*Figure 4: Board Game Arena, showing the 2D interface and platform-imposed rules enforcement*

Further to these examples of general board-gaming platforms, there are several examples of single-game implementations. One of the earliest board games in the modern 'renaissance' era was *The Settlers of Catan* [12], and there have been a number of digital adaptations of this game. The latest of these is *Catan Universe*, available as a stand-alone product for PC, Android, and iOS [13]. As is common with such stand-alone board game implementations, this is a highly polished piece of software. The interface is clear, with a built-in tutorial for the rules of the game and how it is played through the software. The game

components are modelled with a high degree of fidelity, and thought has been given to how the game will display, and be controlled, on devices of varying screen sizes. In this example, the game is modelled fully in 3D, however there is no physics engine implemented, as it is not required for the game to function; components snap to predefined places in the game space, and there is no need to model collisions or forces.


*Figure 5: Catan Universe, showing the 3D rendered environment and clear multiplayer interface*

Other standalone board game implementations follow a similar pattern. I have not been able to find any examples of commercial products adapting a specific dexterity-type board game. Some examples exist of amateur implementations of basic stacking-type games such as *Jenga*. A technology demonstration for the *Physijs* plugin contains a 3D environment with a *Jenga* game set up [14]. This tech-demo has issues preventing it from being a playable game. While collisions and gravity forces are well modelled, controls are rudimentary. Pieces can be selected and dragged, but can only be moved with two degrees of freedom; it is not possible to rotate held pieces, nor to lift them to be placed atop the stack.

Other adaptations of the same game are even more rudimentary. *Jenga* publishers Hasbro released a single-player web version of the game at an indeterminate time in the past. This implementation was run through the Macromedia Flash platform, and is now archived on several websites hosting

abandoned Flash software [15]. This version of the game features no physics simulation, with very simplistic controls and an algorithmic determination of when the stacked objects should collapse.

The increasing use of digital board gaming platforms in recent years was dramatically accelerated by the spread of the COVID-19 pandemic, and the attendant restrictions on social interactions. Online platforms reported dramatic spikes in users, especially in the spring of 2020 [16] [17]. While opportunities for in-person gaming have largely returned to a normal state, the popularity of digital board-gaming has been maintained. New game adaptations are continuously being added to online platforms, and new stand-alone games continue to be published [18]. There is a significant potential user-base for a product in this category.

## 2.3 Physics-based games

The above examples show the state of digital adaptations of board games, and the strengths and issues of various approaches. It is evident that a key consideration for adapting a dexterity game will be the ability for the player to pick up an object and move it with six degrees of freedom – translations along and rotations about three axes. This requirement to move objects around a 3D environment is common in many modern computer games.

*Portal* [19] from Valve Software is a well-known first-person puzzle game. The game makes extensive use of its physics engine with innovative application of momentum conservation through linked, spatially distant 'portals'. In addition to the player avatar traversing the levels, many puzzles require the manipulation of 'weighted storage cubes' to either create platforms for the player to access areas, or to actuate switches. The player is able to pick up and move these cubes (and other objects) fairly intuitively. A single button press grabs an object at the camera focus, so long as it is within a set range. While an object is being held, it is maintained at the camera focus point independent of gravity. The held object is still subject to collisions, and can be used to knock other moveable objects about the game space. As the object is locked to the camera's view, it can be moved using the familiar player movement controls; Y-axis (vertical) movement is limited by the fixed distance from the camera, unless the player finds a way

to move their avatar vertically within the level. Rotation of the held object is more restricted: the held object's rotation with respect to the camera is fixed when the object is picked up. It can be rotated about the Y axis with respect to the game world by moving the player, but other rotation axes would require the object to be dropped and picked up again. The simple and intuitive interface for object manipulation is well executed, and the above-mentioned limitations are irrelevant within the scope of this game. This method would need to be built upon to satisfy the needs of a game focussed on precisely placing objects at arbitrary rotations.

*Garry's Mod* [20], like *Portal*, is built on Valve's Source engine. This software was designed as a general sandbox, and thus has object manipulation as a key feature. As with *Portal*, objects can be picked up and held at the camera focus, and moved within the game environment using the player avatar controls. There are several additional features over the *Portal* implementation which make the system in *Garry's Mod* more useful. Held objects can be moved towards/away from the camera using the mouse scroll wheel. In combination with the camera view angle, this allows much greater flexibility in the Y-axis position of objects. The held object can also be freely rotated, though only about two axes. This is achieved by holding a key which locks the camera view angle and switches the mouse XY movement to controlling rotation of the held object about X and Y axes. Rotation about the Z axis (into the screen) is still not achievable while the object is held.

These additional features should prove useful inspiration for designing an intuitive control scheme for manipulating an object with six degrees of freedom in order to place it precisely with respect to a potentially unstable collection of previously placed objects.

# 3 Design

The product will be a video game playable on a windows PC. The gameplay will be based on the commercial dexterity game *Junk Art* [21], a game revolving around stacking objects with unusual shapes, with the goal of creating the tallest structure, or incorporating the most pieces, without the structure collapsing.

The core gameplay loop will involve each player in turn being randomly assigned an game piece to be added to their structure. The player will have the ability to move and rotate the active piece, before placing onto their structure, ideally without any pieces falling off. As each player's structure grows, adding more pieces will become more challenging, until a collapse occurs. This asynchronous, turn-based approach will allow multiple users to play on the same machine, allowing a multiplayer experience without the need to incorporate online play (which would greatly increase the required development and testing resources).

The game will manifest as a representation of a tabletop holding the game pieces. The player will not have an avatar within this space, but rather will use keyboard and mouse to manoeuvre the camera around the environment, similar to the sandbox environment implemented in *Tabletop Simulator*. These camera controls will by default use WASD keys for translation and mouse XY for rotation. This interface will be familiar to anyone with experience of modern video games, and should be quick to pick up for those without that experience.

When the player selects a game piece to add to their structure, they must be able to manoeuvre it with six degrees of freedom – three dimensions of translation and three axes of rotation. The system employed by *Garry's Mod* will be a basis for this, with additional control to add the final rotational axis missing from that implementation. Translation will follow camera movement: as the player moves and rotates the camera, the held piece will stay at a fixed distance from the camera, held at its centre. The mouse scroll wheel will be used to adjust the distance between piece and camera. By holding an additional command key (e.g. 'Space'), the player can switch to rotation mode; the mouse XY movement and scroll wheel will adjust the piece's rotation within world space, with translation of piece and movement of camera disabled while in this mode.

Aside from the 3D world containing the game pieces, the game window will contain several interface elements which are not part of the physics environment. An initial plan for the layout of an in-progress game is illustrated in Figure 6. Elements will include 'scorecards' with information on the game status of the active player, and each other player, taking inspiration from other stand-alone digital board game adaptations such as *Catan Universe* and *Carcassonne Digital* [22]. This will include points scored in the current game (height of structure) and any penalties accrued from pieces falling from their structure. In a physical game, each player would be randomly assigned a differently shaped component to add to their structure by drawing from a shuffled deck of cards, showing representations of each possible piece. This may be replicated by an interface element, rather than by creating card objects within the 3D environment, which would be needlessly fiddly. Alternatively, a randomised piece may be spawned directly into the player's 'hand'.

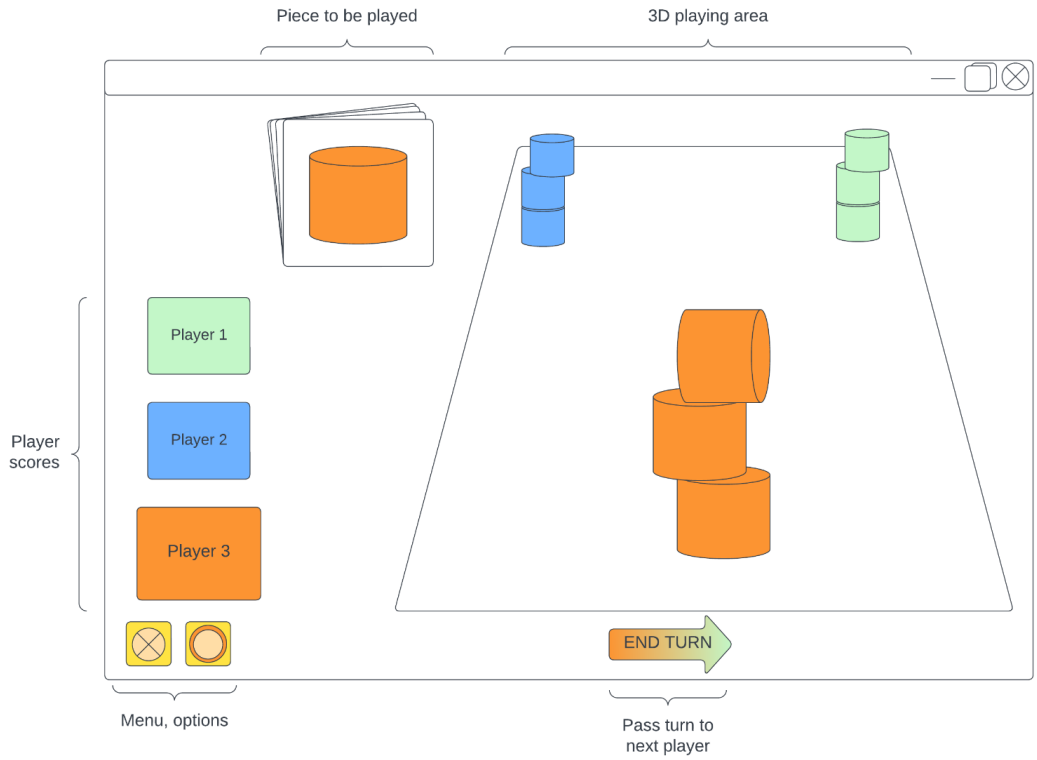


*Figure 6: Game layout*

A button will allow the player to indicate their turn has ended and pass control to the next player. It is possible that this element may not be required, if the game can be configured to detect when the active player has completed all required actions on their turn and automatically advance to the next player; this would not form part of a minimum viable product, but it is likely to create a smoother playing experience and so would be a desirable feature.

The game will need to detect when one or more game pieces fall from any player's structure. Collision detection and object velocity calculations will trigger functions to indicate whether a player has been eliminated due to a collapsed structure. This will also be linked to audio routines to trigger sounds of components falling and striking the tabletop.

The software will be primarily build using the Unity game engine. This development platform allows for creating and configuring the 3D game environment, physics simulation, and 2D interface overlays. It also allows building of the software for multiple platforms, potentially increasing the range of users for both testing and as users of the final product.

Unity includes by default only primitive 3D components. This will be useful for the prototype phase, but to create the unusual shapes which are key for this game, separate 3D modelling software is required. These custom components will be created in Blender. It will be important to create both the render models and also reasonably high-fidelity mesh colliders. Low-vertex box colliders are often used in 3D games to reduce CPU load, however this game revolves around how specific unusual shapes interact, and generic box colliders would remove these different interactions.

The game pieces will be modelled after the components used in the physical game Junk Art, as shown in Figure 7, with a detailed plan of each piece type shown in Figure 8.

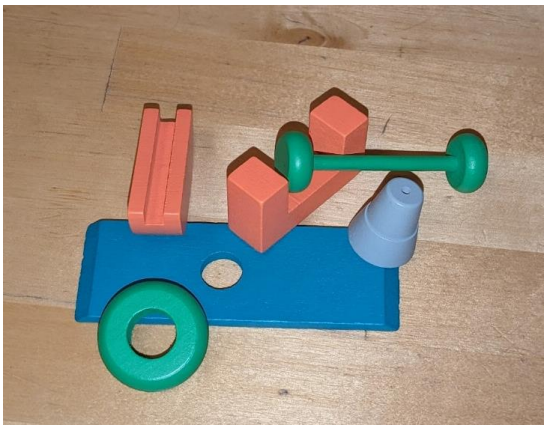

*Figure 7: Physical game pieces*

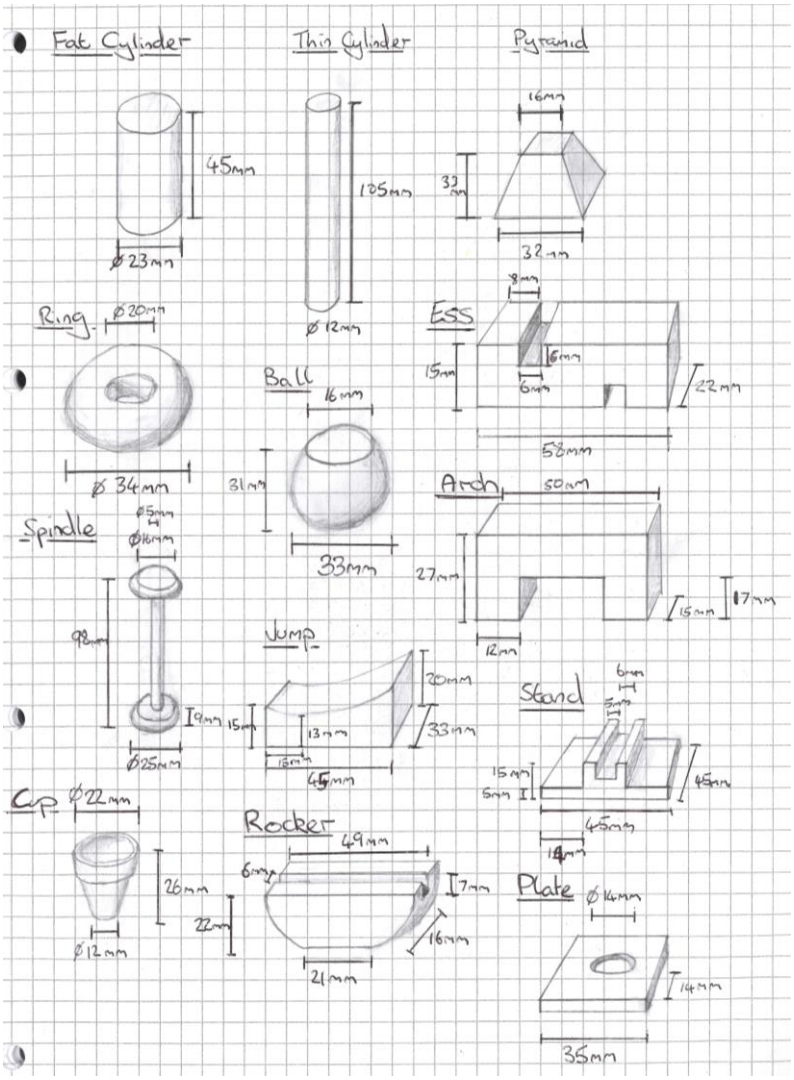


*Figure 8: Game pieces to be included*

2D interface elements will utilise premade assets acquired from the Unity asset store. This resource includes many collections of game UI assets, including free and paid bundles. A free bundle such as MiMU Studio 2D Casual UI [23] will be used for initial prototyping. Depending on the outcome of user testing and feedback, an alternative paid asset set may be required. Possible designs for the scorecard elements are shown in Figure 9.

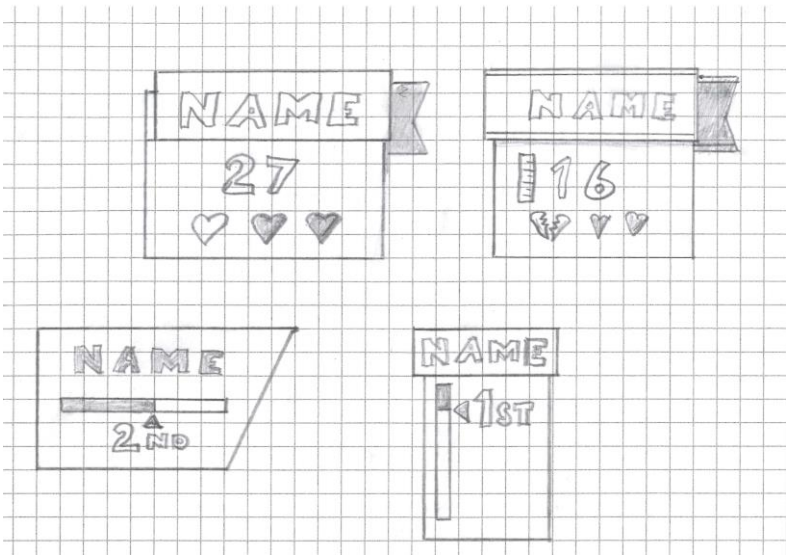


*Figure 9: Scorecard designs*

# 4 Implementation

The game has been created using the Unity game engine. It consistd of two scenes: a main menu, and a game space.

The main menu allows the players to set up and start the game, as well as view information about the game and its controls. A game must have at least two players, and may have up to four. Input fields allow each player's name to be entered, and a colour selected. Functionality of the menu is controlled by the *MainMenu* class.

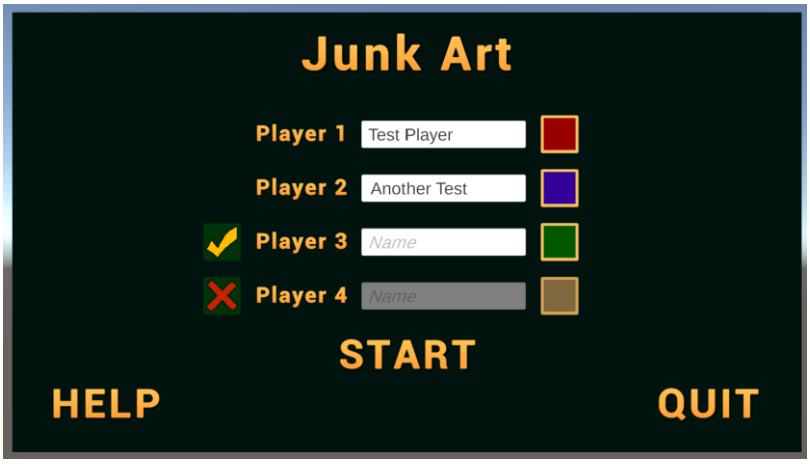


*Figure 10: the main menu, awaiting player 3 name input, with player 4 disabled*

Clicking a colour swatch for a player toggles a colour picker game object, with buttons to select a colour for the player.

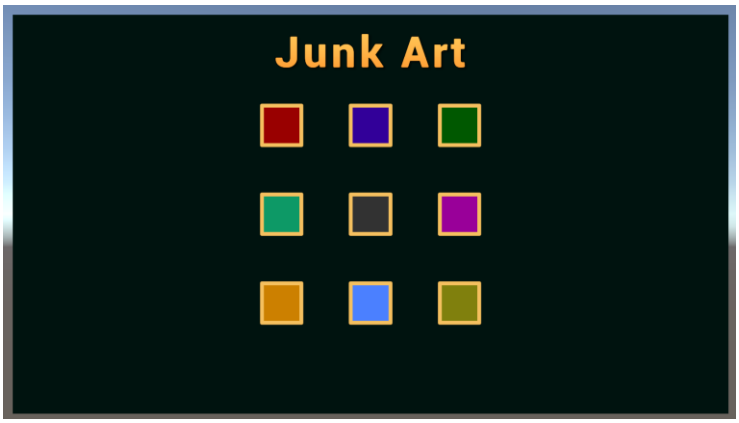


*Figure 11: the colour picker*

Functions within the *MainMenu* class control the toggling of the picker object and the recording of the chosen colour.

```
/// <summary>
/// Activate the color picker for the chosen player
/// </summary>
/// <param name="player">The player geing assinged a new color</param>
0 references
public void PickColor(int player)
{
    //record which player we're picking for
    colorPickPlayer = player;

    //enable the picker
    transform.parent.Find("ColorPicker").gameObject.SetActive(true);
}

/// <summary>
/// Update a player's chosen color, from a swatch image
/// </summary>
/// <param name="swatch">The image containing the chosen color</param>
0 references
public void AssignColor(Image swatch)
{
    //get color
    Color chosenColor = swatch.GetComponent<Image>().color;

    //save to settings
    if (colorPickPlayer == 1) GameSettings.Player1_color = chosenColor;
    if (colorPickPlayer == 2) GameSettings.Player2_color = chosenColor;
    if (colorPickPlayer == 3) GameSettings.Player3_color = chosenColor;
    if (colorPickPlayer == 4) GameSettings.Player4_color = chosenColor;

    //disable picker
    transform.parent.Find("ColorPicker").gameObject.SetActive(false);

    //update displayed colour for all players
    UpdatePlayerColors();
}
```

*Figure 12: functions for selecting a colour for a player*

The colours, along with the player names and activation status are stored to a static class *GameSettings*. The static nature ensures there is only a single instance of the class, and it can be referenced solely by the class name.

Clicking the START button within the main menu causes the game to launch with the settings provided.

The world loaded by the game scene consists initially of only a table (from the Unity asset store [24]). Components for the game are instantiated by the *GameController* class. On loading the scene, the *Start* function of this class carries out several tasks:

- Create an array of the appropriate size to hold *Player* objects
- Create a shuffled 'deck' of game pieces to spawn on each turn
- Create *Player* objects
- Start the first player's turn

```
Unity Message | 0 references
private void Start()
{
    SetupPlayers(); //get player count and fill arrays

    deck = ScriptableObject.CreateInstance<GamePieceDeck>(); //instantiate a deck of pieces
    deck.InitDeck(playerCount); //populate and shuffle the deck

    CreatePlayers();

    //start the game
    gameOver = false;
    activePlayer = -1;
    NextPlayer();
}
```

*Figure 13: functions called on game start*

The *Player* class, from which the player objects are instantiated, holds details of the player's number, name and colour - taken from the *GameSettings* class. It also contains functions to create a 'structure base' game object in the world, and a scorecard in the user interface (UI). The positions of these are calculated by the *GameController* class and passed in by the *CreatePlayers* function.

```
/// <summary>
/// Calculate player base position by dividing circular table evenly by player count
/// </summary>
/// <param name="pCount">Total number of player in this game</param>
/// <param name="pNum">This player's number</param>
/// <returns>The position of the player base</returns>
1 reference
private Vector3 PlayerHomePos(int pCount, int pNum)
{
    float angle = (2f * Mathf.PI / pCount) * pNum; //radians around circle

    float y = 0f;
    float x = Mathf.Cos(angle) * radius;
    float z = Mathf.Sin(angle) * radius;

    return new Vector3(x, y, z);
}
```

*Figure 14: Calculation of a player's structure base position on the game table*

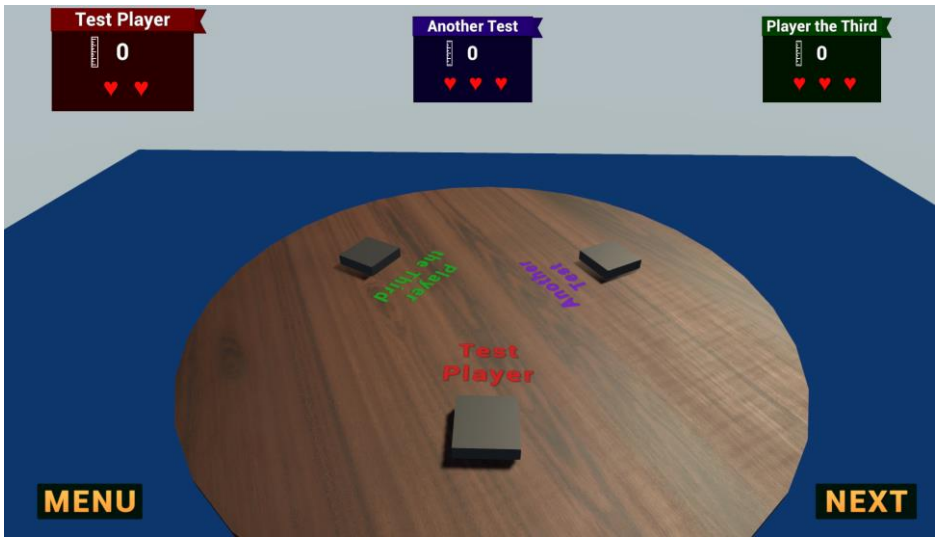


*Figure 15: the game table set up for three players, with bases and scorecards spawned*

The player does not require an avatar within the game world, so the camera is controlled directly using the WASD-and-mouse scheme familiar to players of modern games. This is implemented in the *CameraController* class attached to the main camera. This script is adapted from an example included one of the template projects bundled with Unity.

An additional class *GrabObject* is also attached to the main camera. This class controls the behaviour of a game piece while being 'held' by the player. Game pieces are spawned in the 'held' state by the *GameController*, and the *GrabObject* class ensures that the 'held' pieces are kept at a focus point in front of the camera as it moves around the game world.

```
/// <summary>
/// Move held object towards main camera descendent
/// </summary>
1 reference
private void MoveObject()
{
    if (Vector3.Distance(heldObject.transform.position, holdFocus.position) > 0.1f)
    {
        //vector from object to hold point
        Vector3 moveDirection = holdFocus.position - heldObject.transform.position;

        //move using velocity
        heldObjectRB.velocity = moveDirection * moveDirection.magnitude * grabForce;
    }
}
```

*Figure 16: Keeping a held object at the camera focus position*

The player can use the mouse scroll wheel to move the held piece towards or away from the camera. Together with the camera movement functions, this allows the player to translate the held object to any point in the game world: three degrees of freedom.

```
//restrict max scrolling
scrollDistance = Mathf.Clamp(scrollDistance + Input.mouseScrollDelta.y * scrollSpeed, -maxScroll, maxScroll);
//move hold position when scrolling
holdFocus.position = grabFocus.position + (holdFocus.TransformDirection(Vector3.forward) * scrollDistance);
```

*Figure 17: Moving the location of the held object to or from the camera with the mouse scroll wheel*

The three remaining degrees of freedom specified in the project design are the rotations of the held game piece around its three axes. When the Space key is held the *GrabObject* class does two things: an event is invoked to call a function on the *CameraController,* halting interpretation of inputs to move the main camera. Instead the *GrabObject* class takes the two axes of mouse movement plus the scroll wheel and applies them as rotations to the game piece.

```
/// <summary>
/// Rotate held object on 3 axes according to mouse movement and scrolling
/// </summary>
1 reference
private void RotateObject()
{
    //get mouse movement
    Vector3 mouseMovement = new Vector3(
        Input.GetAxis("Mouse Y") * -1,
        Input.GetAxis("Mouse X") * -1,
        Input.mouseScrollDelta.y * 5
        );

    //rotate the object
    heldObjectRB.transform.Rotate(mouseMovement);
}
```

*Figure 18: Rotation of a held game piece*

The game pieces were created based on the design document using the *Blender* modelling software. Due to their comparatively simple, geometric nature, it was possible to create all pieces by manual vertex manipulation, with vertex subdivision smoothing and edge splitting where required.

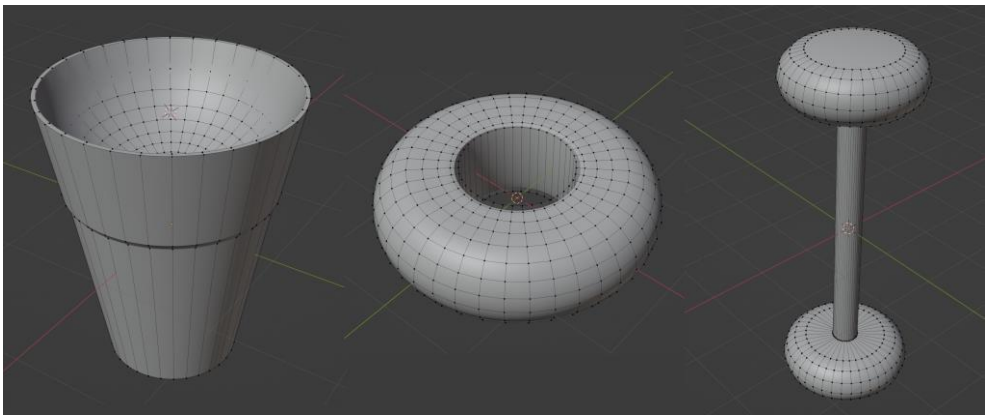

*Figure 19: Blender models of three game pieces: 'cup', 'ring' ,and 'spindle'*

Realistic physics behaviour of stacking the game pieces requires each piece to have a collider closely confirming to the model shape. For some shapes a simple convex mesh collider could be automatically computed by Unity.

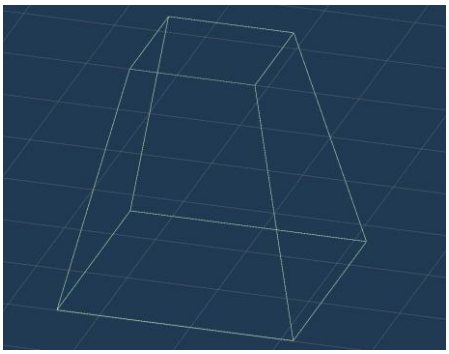

*Figure 20: The 'pyramid' game piece mesh collider*

Pieces which do not have a convex hull could not be dealt with this way, as each collider must be convex to allow physics modelling. Some of the game pieces could be modelled by combining several box colliders.

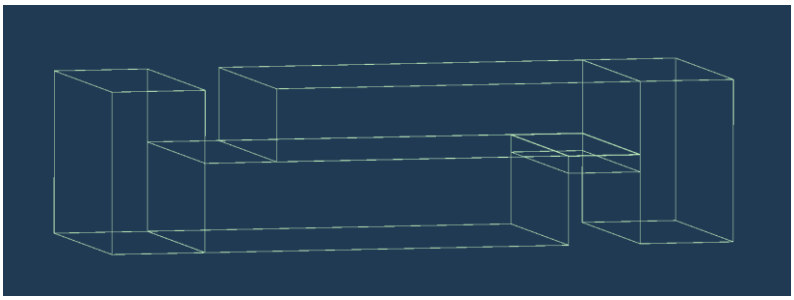

*Figure 21: The five non-overlapping box colliders for the 'ess' game piece*

Some of the game pieces, such as the 'ring' shown in Figure 19 required another approach. In order to accurately model non-convex shapes with curved hulls, the model mesh needed to be decomposed into a finite number of complex, convex, non-overlapping colliders. To achieve this, the Voxelised Hierarchical Convex Decomposition (V-HACD) algorithm was applied. This process was developed by John Ratcliff, with contributions from Khaled Mamou [25] [26]. The algorithm converts a model's mesh into a group of voxels and recursively divides this group into subgroups until the volume of a bounding box around the subgroup approximates the volume of the voxels within, to within a given tolerance. These bounding boxes (of which there will likely be thousands) can then be merged back up to the required number of final colliders, while maintaining a convex shape for each, and minimising any extra volume enclosed by the merged boxes.

The V-HACD process is provided as a C++ library and application under the BSD 3-Clause license [27]. Running the application against the model files exported from blender provided a set of colliders which can be imported into Unity along with the model. V-HACD tuning parameters for each game piece were selected by trial-and-error to produce a set of colliders for each piece that maximised fidelity to the shape without risking extraneous runtime resource requirements for the final programme. For example, the 'ring' piece shown in Figure 22 was produced from 14 colliders with a maximum of 40 vertices each.

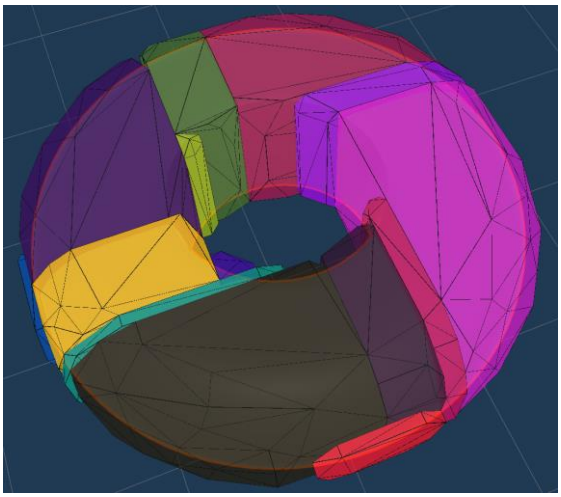

*Figure 22: Colliders for the 'ring' piece generated by the V-HACD algorithm*

The game proceeds with each player taking a turn, on which a random game piece is spawned for the player to add to their structure. Each piece must be placed atop either the player's 'base' or balanced atop other pieces already in the player's growing structure. If any piece touches the game table, or the floor, it removed from the game. This is accomplished by a collision handling function in the *GamePiece* class attached to each game piece. When a collision occurs with an object tagged as 'ground', its *destroy* property is set to true, which causes the *Update* loop to execute a short countdown until the piece is removed. While the countdown is running, the renderer component of the game piece is toggled on and off, causing the piece to 'flash' in the game.

```
Unity Message | 0 references
void Update()
{
    if (destroy)
    {
        FlashAndDestroy();
    }
}

/// <summary>
/// Called once per update loop
/// Flash this object while awating destruction by dis/enabling renderer
/// At end of destruction time destroy this object
/// </summary>
1 reference
private void FlashAndDestroy()
{
    //event triggers at start of destruction timer
    if (destroyTimer == 0f)
    {
        onDestruction?.Invoke(OwnerIndex);
    }

    //increment counters
    destroyTimer += Time.deltaTime;
    flashTimer += Time.deltaTime;

    //destroy at end of total timer
    if (destroyTimer >= detroyLen)
    {
        Destroy(gameObject);
    }

    //alternate renderer and reset flash timer
    if (flashTimer >= flashLen)
    {
        thisRenderer.enabled = !thisRenderer.enabled;
        flashTimer = 0f;
    }
}
```

*Figure 23: Destruction of a game piece*

The *FlashAndDestroy* function also invokes an event to call a function on the *GameController* which decrements the life counter on the *Player* object for the owner of the destroyed piece.

Scoring is determined by the height above the table of each player's structure. This is determined by looping through each game piece in all stacks and finding the maximum Y value of the collider's axis-aligned bounding box (AABB). The nature of the AABB means this value will always be the highest point on of the piece's collider within the world space. For game pieces with composite physics colliders, a separate trigger-only box collider is used for this purpose. For each player, the greatest Y bound of the AABBs for their stacked pieces is taken to be their base score.

```
/// <summary>
/// Calculate maximum height for each player's stack
/// Update the player objects with that value
/// </summary>
3 references
private void StackHeights()
{
    //do nothing after game ends
    if (gameOver) return;

    //make a hash table for the stacks
    //values default to 0f
    float[] stackHeights = new float[playerCount];

    GameObject[] pieceList = GameObject.FindGameObjectsWithTag("GamePiece");

    //loop through each peice in play
    foreach(GameObject piece in pieceList)
    {
        //piece still held, don't count it, move on
        if (piece.GetComponent<GamePiece>().Held)...

        //piece moving (falling), don't count it, move on
        if (piece.GetComponent<Rigidbody>().velocity.magnitude > 0.6)...

        //owner of stack
        int pieceOwner = piece.GetComponent<GamePiece>().OwnerIndex;

        //piece not owned, don't count it, move on
        if (pieceOwner < 0)...

        //top of axis-aligned bounding box
        float pieceHeight = piece.GetComponent<Collider>().bounds.max.y;

        //update stack heights if greater
        if (pieceHeight > stackHeights[pieceOwner])
        {
            stackHeights[pieceOwner] = pieceHeight;
        }
    }

    //write the heights to the player objects
    for(int i = 0; i < stackHeights.Length; i++)
    {
        playerArray[i].UpdateScore(stackHeights[i]);
    }
}
```

*Figure 24: calculating each player's structure height*

A score penalty is applied for each life lost to give the player's current score. This value is displayed in each player's score card, as is their current life count.

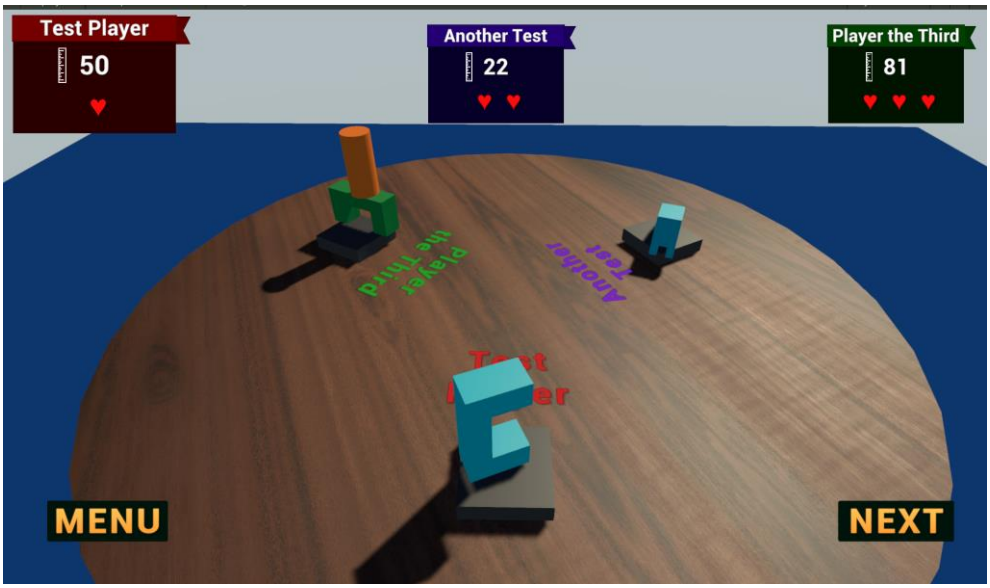


*Figure 25: Game in progress, showing scores and lives lost in the scorecards*

Each player starts with three lives. When any player reaches 0 lives, the game ends and each player's final score is displayed.

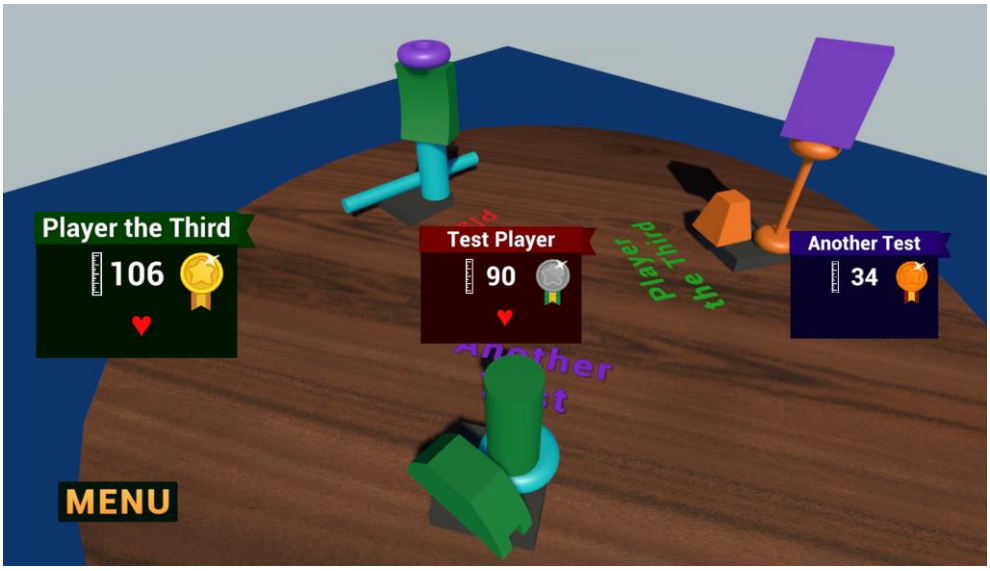


*Figure 26: Game over and final scores*

# 5 Evaluation

The initial aims of this project were defined in combination by the provided project template, and by the outcome of the discovery phase. Those objectives can be broadly split into two categories: objective or functional, and subjective.

Objective or functional aims include:

- The game functions well without crashing
- The game uses physics simulation as a key element of gameplay
- The game follows the core gameplay loop of the physical game 'Junk Art'
- The player is able to manipulate game objects with six degrees of freedom

Subjective aims include:

- The game is easy to understand
- The game is easy to control and play
- The game is fun and gratifying

These categories required different means of evaluation. Objective aims could be primarily evaluated by the developer, using a set of test cases against which the different iterations of the software were evaluated. These test cases are enumerated in Appendix 7.1. Ensuring that all tests were carried out after each major addition or revision to the software ensured its stability, and adherence to both the requirements, and the project design. Later in the development cycle, after some external testers had been recruited and had experienced several rounds of subjective testing (as described below), they were also asked to run through these test cases, both to ensure the objective aims were being met, and to ensure that these external testers had experience of, and could given feedback on, all aspects of the software. Issues or bugs exposed by this testing were investigated and resolved on a case-by-case basis, so there is no qualitative or quantitative data resulting from the testing of these objectives.

To do: Evaluation of subjective aims section

- Subjective testing
    - Early user playtesting
        - Incomplete product
        - Supervised playtesting
        - User interviews for feedback
        - Results and suggestions incorporated into next iteration design where feasible
            - Choose player colours
            - Distinguishing 'held' objects: outline highlight shader
            - Sound effects on collisions
            - Bigger starting bases to allow for increased difficulty of object placement compared to IRL game
    - Final user playtesting (not yet undertaken)
        - Complete product
        - Unsupervised playtesting
        - Survey with qualitative & quantitative questions
            - Questions in Appendix 7.2
        - Results to be discussed
        - Suggestions as potential future work

# 6 Conclusions

To do:

- Which objectives and user needs have been met
  - Game runs
  - Successfully emulates gameplay of a dexterity board game based on stacking pieces
  - Core gameplay loop is short
  - Playable by 2-4 players in hot-seat mode
- Which objectives and user needs have not been met?
  - Boring to play
  - Frustrating to achieve intended results due to inability to fully judge position of pieces relative to each other
- How could the implementation be improved?
  - Player should be able to configure controls
  - Background music
  - More sound effects
  - More clarity on rotation axes of game pieces
- How does the process of implementing and evaluating the project change the view of the initial objectives?
  - Digital adaptations of tabletop board games are enjoyed by many people (ref literature review)
  - Digital adaptations of dexterity-type board games fail to capture the fun of playing those IRL
  - Digitising the experience removes the point of dexterity games, unlike other board games where the mental challenge remains
  - Stacking games are especially troublesome to digitise: they rely on full awareness of game component positions in 3D space, which is not feasible through a 2D screen
    - Maybe would work better in VR?
  - Refer to dearth of extant digital adaptations of dexterity games as proof of this fundamental flaw, rather than a gap in the market

# 7 Appendices

## 7.1 Test cases

| Test Name | Game loads |
|---|---|
| Preconditions | None |
| Test Steps | Open software |
| Expected Results | Main menu displayed |

| Test Name | Main menu: player names |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click on player name field. Type player name |
| Expected Results | Player name can be typed into field |

| Test Name | Main menu: change player colour |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click on colour button for a player |
| Expected Results | Colour picker screen displayed with nine preset options |

| Test Name | Main menu: pick player colour |
|---|---|
| Preconditions | Colour picker screen displayed |
| Test Steps | Click a colour option |
| Expected Results | Main menu displayed.<br>Chosen colour displayed next to relevant player |

| Test Name | Main menu: deactivate player |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click ‘tick’ button next to player 3 or 4 |
| Expected Results | ‘Tick’ changes to ‘cross’.<br>Player name field and player colour button are greyed out and not interactable |

| Test Name | Main menu: reactivate player |
|---|---|
| Preconditions | Main menu displayed<br>Player 3 or 4 deactivated |
| Test Steps | Click ‘cross’ button next to player 3 or 4 |
| Expected Results | ‘Cross changes to ‘tick.<br>Player name field and player colour button are no longer greyed out and now interactable |

| Test Name | Main menu: activation cascade |
|---|---|
| Preconditions | Main menu displayed<br>Player 3 and 4 both active |

| Test Steps | Click ‘cross’ button next to player 3 |
|---|---|
| Expected Results | Player 4 deactivates as well as player 3 |

| Test Name | Main menu: reactivation cascade |
|---|---|
| Preconditions | Main menu displayed<br>Player 3 or 4 both inactive |
| Test Steps | Click ‘cross’ button next to player 3 |
| Expected Results | Player 4 reactivates as well as player 3 |

| Test Name | Main menu: user help |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click HELP |
| Expected Results | User help screen displayed, with game information and controls |

| Test Name | Main menu: close user help |
|---|---|
| Preconditions | User help screen displayed |
| Test Steps | Click BACK |
| Expected Results | Main menu displayed, with any player choices maintained. |

| Test Name | Main menu: exit game |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click QUIT |
| Expected Results | Application closes |

| Test Name | Main menu: start game |
|---|---|
| Preconditions | Main menu displayed |
| Test Steps | Click START |
| Expected Results | Game table displayed |

| Test Name | Scorecards: count |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards displayed across top of screen for each active player |

| Test Name | Scorecards: names |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards show correct player names |

| Test Name | Scorecards: default names |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards show ‘Player #’ if no name was entered |

| Test Name | Scorecards: colours |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards show chosen player colours |

| Test Name | Scorecards: lives |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards show three lives for each player |

| Test Name | Scorecards: scores |
|---|---|
| Preconditions | Game started |
| Expected Results | Scorecards show each player with 0 points |

| Test Name | Scorecards: active player |
|---|---|
| Preconditions | Game started |
| Expected Results | Player 1 scorecard is shown larger |

| Test Name | Structure bases: count |
|---|---|
| Preconditions | Game started |
| Expected Results | Bases created on table for each player |

| Test Name | Structure bases: position |
|---|---|
| Preconditions | Game started |
| Expected Results | Bases spaced equally around circular table |

| Test Name | Structure bases: names |
|---|---|
| Preconditions | Game started |
| Expected Results | Correct player names shown above each base |

| Test Name | Structure bases: default names |
|---|---|
| Preconditions | Game started |
| Expected Results | ‘Player #’ shown above each base if no name entered |

| Test Name | Structure bases: colour |
|---|---|
| Preconditions | Game started |
| Expected Results | Player names shown in chosen player colours |

| Test Name | Gameplay: initial camera position |
|---|---|
| Preconditions | Game started |
| Expected Results | Camera positioned to view game table<br>Camera centred behind Player 1 base |

| Test Name | Gameplay: piece spawned |
|---|---|

| | |
|---|---|
| Preconditions | Game started |
| Expected Results | Game piece spawned in held state<br>Spawned piece highlighted with yellow edge glow |

| | |
|---|---|
| Test Name | Gameplay: camera horizontal movement |
| Preconditions | Game started |
| Test Steps | Use WASD keys to move camera around game world |
| Expected Results | Camera translates forward, backward, left, right |

| | |
|---|---|
| Test Name | Gameplay: camera vertical movement |
| Preconditions | Game started |
| Test Steps | Use Q and E keys to move camera around game world |
| Expected Results | Camera translates up and down |

| | |
|---|---|
| Test Name | Gameplay: camera rotation |
| Preconditions | Game started |
| Test Steps | Hold right mouse button and move mouse |
| Expected Results | Camera rotates with mouse movement |

| | |
|---|---|
| Test Name | Gameplay: game piece translation with camera |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Use camera controls to move camera |
| Expected Results | Game piece is held at camera focus<br>Game piece moves with camera |

| | |
|---|---|
| Test Name | Gameplay: game piece translation to / from camera |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Use mouse scroll wheel |
| Expected Results | Game piece moves towards/away from camera<br>Game piece will not move too close or far from camera |

| | |
|---|---|
| Test Name | Gameplay: game piece rotation |
| Preconditions | Game piece held |
| Test Steps | Hold Space bar<br>Move mouse horizontally<br>Move mouse vertically<br>Use mouse scroll wheel |
| Expected Results | Game piece rotates around three axes |

| | |
|---|---|
| Test Name | Gameplay: camera lock in rotation mode |
| Preconditions | Game piece held |

| | |
|---|---|
| Test Steps | Hold Space bar<br>Move mouse horizontally<br>Move mouse vertically<br>Use mouse scroll wheel |
| Expected Results | Camera does not move around game world<br>Game piece does not move to/from camera |

| | |
|---|---|
| Test Name | Gameplay: drop game piece |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Click left mouse button |
| Expected Results | Held game piece dropped<br>Game piece falls under gravity<br>Game piece no longer highlighted with yellow outline |

| | |
|---|---|
| Test Name | Gameplay: piece lands on base |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Drop game piece to land on structure base |
| Expected Results | Game piece lands on structure base<br>Game piece not destroyed |

| | |
|---|---|
| Test Name | Gameplay: piece lands on another piece |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Drop game piece to land on another game piece |
| Expected Results | Game piece lands on structure base<br>Game piece not destroyed |

| | |
|---|---|
| Test Name | Gameplay: piece lands on table |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Drop game piece to land touching the table |
| Expected Results | Game piece lands touching the table<br>Game piece flashes for 1.5 seconds<br>Game piece destroyed after 1.5 seconds |

| | |
|---|---|
| Test Name | Gameplay: life lost from held piece |
| Preconditions | Game started<br>Game piece held |
| Test Steps | Drop game piece to land touching the table |
| Expected Results | Active player loses 1 life<br>Active player score updates with penalty applied |

| Test Name | Gameplay: life lost from stacked piece |
|---|---|
| Preconditions | Game started<br>Game pieces stacked on player bases |
| Test Steps | Cause game pieces to fall to table from stack |
| Expected Results | Piece owner loses 1 life<br>Piece owner score updates with penalty applied |

| Test Name | Gameplay: next player |
|---|---|
| Preconditions | Game started |
| Test Steps | Click NEXT |
| Expected Results | Camera realigns to view next player's structure<br>Previous player's scorecard reverts to normal size<br>Next player's scorecard enlarges<br>Players' scores updated based on structure heights |

| Test Name | Gameplay: player loop |
|---|---|
| Preconditions | Game started<br>Last player is active |
| Test Steps | Click NEXT |
| Expected Results | First player becomes active |

| Test Name | Gameplay: open pause menu |
|---|---|
| Preconditions | Game started |
| Test Steps | Click MENU |
| Expected Results | Pause menu displayed<br>Camera locked |

| Test Name | Gameplay: close pause menu |
|---|---|
| Preconditions | Game started<br>Pause menu opened |
| Test Steps | Click BACK |
| Expected Results | Pause menu closed<br>Camera unlocked |

| Test Name | Gameplay: return to main menu |
|---|---|
| Preconditions | Game started<br>Pause menu opened |
| Test Steps | Click MAIN MENU |
| Expected Results | Main menu displayed with default settings |

| Test Name | Gameplay: exit game |
|---|---|
| Preconditions | Game started |

| | |
|---|---|
| | Pause menu opened |
| Test Steps | Click QUIT |
| Expected Results | Application closes |

| Test Name | Gameplay: game over |
|---|---|
| Preconditions | Game started |
| Test Steps | Cause one player to lose three lives |
| Expected Results | Camera locked<br>Scorecards displayed across centre of screen<br>Scorecards sorted by final score<br>Gold, silver, bronze rosettes displayed on scorecards |

## 7.2 User survey questions

# Junk Art - playtest survey

Thank you for agreeing to playtest this digital adaptation of Junk Art. The following 12 questions will help me improve the game.

Were you able to run the game application?

- ◯ Yes
- ◯ No

How easy was it to understand the object of the game?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| I could not understand | ◯ | ◯ | ◯ | ◯ | ◯ | I understood immediately |

How easy was it to move the camera in the game world?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| I could not move the camera | ◯ | ◯ | ◯ | ◯ | ◯ | I could easily place the camera where I wanted |

How easy was it to place the game pieces where you intended?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| I could not move the game pieces at all | ◯ | ◯ | ◯ | ◯ | ◯ | I could easily place the pieces where I wanted |

How understandable was the scoring system?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| I did not understand how the game was scored | ◯ | ◯ | ◯ | ◯ | ◯ | I immediately understood how the game was scored |

How challenging was the gameplay?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| The game was far too easy to play | ○ | ○ | ○ | ○ | ○ | The game was far too hard to play |

How fun did you find the game?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| The game was very dull | ○ | ○ | ○ | ○ | ○ | The game was very engaging |

How long did you play for?

- ○ Less than 2 minutes
- ○ 2-5 minutes
- ○ 5-10 minutes
- ○ 10-20 minutes
- ○ More than 20 minutes

How often do you play video games?

- ○ Never
- ○ A few times a year
- ○ A few times a month
- ○ A few times a week
- ○ Every day

How often do you play tabletop board games?

- ◯ Never
- ◯ A few times a year
- ◯ A few times a month
- ◯ A few times a week
- ◯ Every day

What did you find most frustrating about the game?

Your answer

What was your favourite thing about the game?

Your answer

# 8 References


[1] P. Shotwell, "Some New Approaches to the Study of the History of Go in Ancient China and Siberia," in *The 2nd International Conference on Baduk: proceedings*, Seoul, 2003.

[2] C. L. Wooley, Ur Excavations Volume II: The Royal Cemetery, London: Publications of the Joint Expeditions of the British Museum and the Museum of the University of Pennsylvania, 1934.

[3] M. Sebbane, "Board Games from Canaan in the Early and Intermediate Bronze Ages and the Origin of the Egyptian Senet Game," *Tel Aviv,* vol. 28, no. 2, pp. 213-230, 2013.

[4] M. Hofer, The Games We Played: The Golden Age of Board & Table Games, New York: Princeton Architectural Press, 2003.

[5] A. Seville, "Early history and meaning of the Game of the Goose," in *The Cultural Legacy of the Royal Game of the Goose: 400 years of Printed Board Games*, Amsterdam, Amsterdam University Press, 2019, pp. 22-36.

[6] R. Kennedy Jr. and J. Waltzer, "The Game - A Brief History of Monopoly," in *Monopoly: The Story Behind the World's Best Selling Game*, Layton, UT, Gibbs Smith, 2004, pp. 8-14.

[7] S. Woods, Eurogames: The Design, Culture and Play of Modern European Board Games, Jefferson N.C.: McFarland & Company, 2012.

[8] P. Konieczny, "Golden Age of Tabletop Gaming: Creation of the Social Capital and Rise of Third Spaces for Tabletop Gaming in the 21st Century," *Polish Sociological Review,* vol. 206, no. 2, pp. 199-215, 2019.

[9] Berserk Games LLC, "Tabletop Simulator," [Online]. Available: https://www.tabletopsimulator.com/. [Accessed 26 10 2023].

[10] Berserk Games LLC, "Tabletop Simulator Knowledge Base: Controls & Movement," 17 04 2021. [Online]. Available: https://kb.tabletopsimulator.com/player-guides/basic-controls/. [Accessed 10 11 2023].

[11] AD2G Studio SAS, “Board Game Arena,” [Online]. Available: https://boardgamearena.com/. [Accessed 26 10 2023].

[12] L. Levy, “Special K - Klaus Teuber,” 08 2001. [Online]. Available: https://web.archive.org/web/20160131095945/http://www.thegamesjournal.com/articles/SpecialK3.shtml . [Accessed 16 11 2023].

[13] KOSMOS Publishing, “Catan Universe,” [Online]. Available: https://catanuniverse.com/en/. [Accessed 16 11 2023].

[14] C. Prall, “Jenga - Physijs,” 19 10 2015. [Online]. Available: https://chandlerprall.github.io/Physijs/examples/jenga.html. [Accessed 16 11 2023].

[15] “Jenga,” Gamenora, [Online]. Available: https://www.gamenora.com/game/jenga/. [Accessed 16 11 2023].

[16] A. Lauer, “Virtual Board Gaming Is Your Unlikely Quarantine Savior,” Inside Hook, 30 03 2020. [Online]. Available: https://www.insidehook.com/culture/play-online-board-games-during-quarantine. [Accessed 21 10 2023].

[17] Berserk Games LLC, “The Future of Virtual Gaming One Year Later After Covid,” 19 03 2021. [Online]. Available: The Future of Virtual Gaming One Year Later After Covid. [Accessed 20 10 2023].

[18] Board Game Arena, “Summer of Games 2023: a full month of daily releases!,” 31 07 2023. [Online]. Available: https://boardgamearena.com/news?f=10&t=31756&s=SUMMER+of+GAMES+2023%3A+a+full+month+of+daily+releases. [Accessed 22 10 2023].

[19] Valve Corporation, “Portal,” 10 10 2007. [Online]. Available: https://store.steampowered.com/app/400/Portal/. [Accessed 17 11 2023].

[20] Facepunch Studios, “Garry's Mod,” 29 11 2006. [Online]. Available: https://gmod.facepunch.com/. [Accessed 17 11 2023].

[21] Asmodee, “Junk Art 3.0,” [Online]. Available: https://www.asmodee.co.uk/products/pbgpzg20030_junk-art-3-0. [Accessed 25 10 2023].

[22] Asmedee Digital, "Carcassonne," 2019. [Online]. Available: https://www.asmodee-digital.com/en/carcassonne/. [Accessed 25 11 2023].

[23] "2D Casual UI HD," MiMU Studio, 23 02 2017. [Online]. Available: https://assetstore.unity.com/packages/2d/gui/icons/2d-casual-ui-hd-82080. [Accessed 26 11 2023].

[24] L. Szymanska, "Unity Asset Store: Dinning Set," 23 02 2021. [Online]. Available: https://assetstore.unity.com/packages/3d/props/furniture/dinning-set-186476. [Accessed 28 12 2023].

[25] K. Mamou, "Volumetric Hierarchical Approximate Convex Decomposition," in *Game Engine Gems 3*, Boca Raton, FL, Taylor Francis, 2016, pp. 141-158.

[26] J. Ratcliff and K. Mamou, "Voxelized Hierarchical Convex Decomposition - V-HACD version 4," 04 2022. [Online]. Available: https://docs.google.com/presentation/d/1OZ4mtZYrGEC8qffqb8F7Le2xzufiqvaPpRbLHKKgTIM/edit#slide=id.p. [Accessed 28 12 2023].

[27] J. Ratcliff and K. Mamou, "GitHub - Voxelized Hierarchical Convex Decomposition," 14 12 2023. [Online]. Available: https://github.com/kmammou/v-hacd. [Accessed 28 12 2023].