Junk Art

A digital board game adaptation

Preliminary report

Thomas Nagel

December 2023

Content

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].

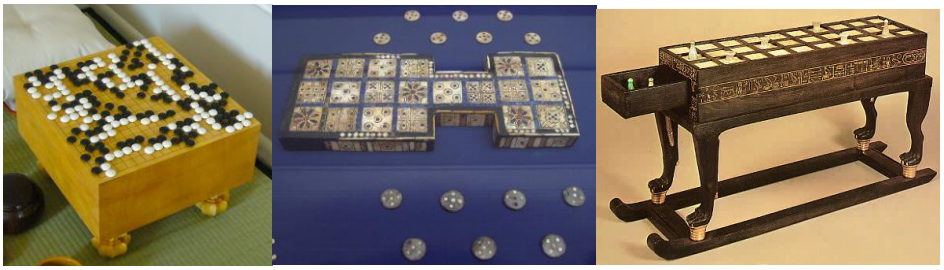


Figure 1: [L-R] Go, Royal Game of Ur, and Senet board games

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 core 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 unused 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 having familiarity with 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. 'E'), 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. This combined control aspect will need careful testing and iterative development to ensure players can easily manoeuvre pieces as they require, in order to place then onto their structures.

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 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 each round of the game, and progress in the current round (height of structure etc.). 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 will be replicated by an interface element, rather than by creating card objects within the 3D environment, which would be needlessly fiddly.

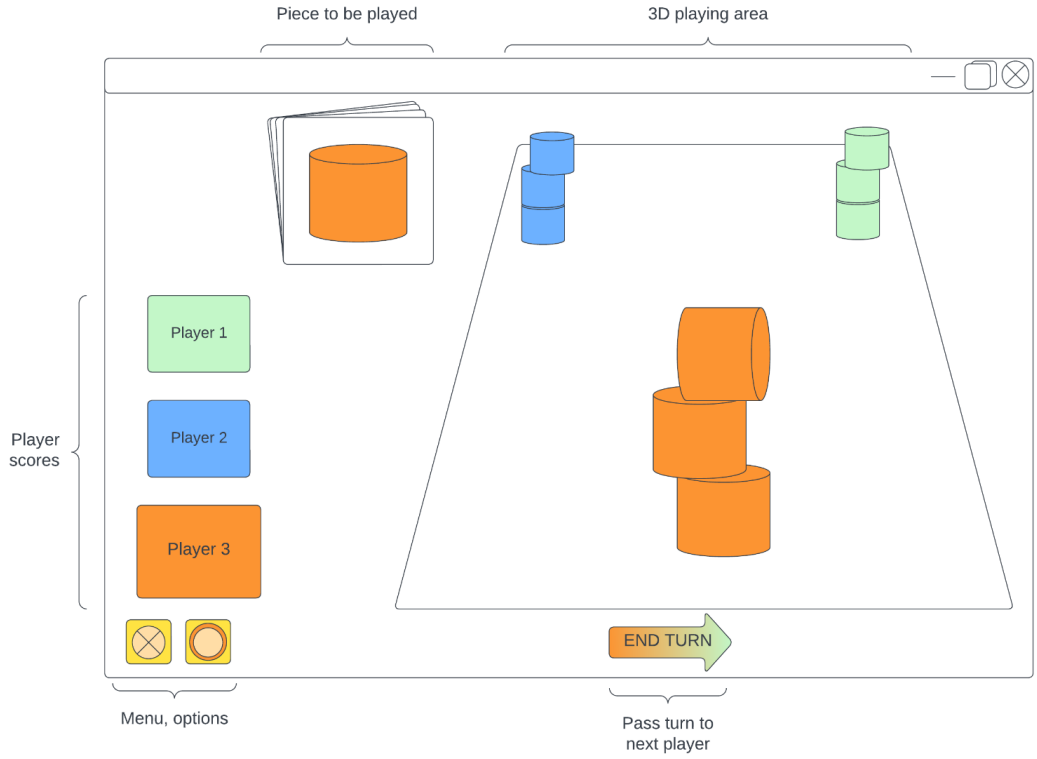


Figure 6: Game layout

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. Examples of game pieces from the physical game Junk Art are shown in Figure 7. The digital adaptation for this project will not necessarily replicate these pieces, but these will provide inspiration for components created within the Blender software.

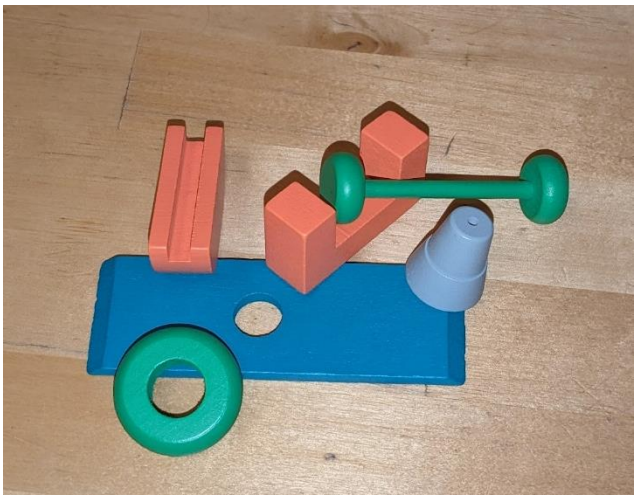


Figure 7: Physical game pieces

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.

4 Work plan

The work plan for this project is illustrated in the Gantt chart in Appendix 7.1. This chart shows the high-level objectives for the planning and development phases of the project, with allowances for break times over the Christmas and New Year periods as well as a pre-arranged holiday.

During the development phase, there will be several build points to allow for user testing of the features implemented to that point. It is intended that this testing will continue while the next stage of development continues. A large amount of time has been allocated to this testing, as it will rely on volunteers who may only be able to dedicate a small amount of their time during the allocated testing periods. Changes and enhancements resulting from user feedback will be incorporated during the entire development phase, with a final week (labelled 'refinement') dedicated to any last changes indicated from the final round of user testing.

5 Evaluation plan

The aims for this project are derived from a combination of the provided template and the specific design chosen. The final software should be a game that is:

- Stable and bug free
- Easy to pick up and play
- Fun to play
- A good adaptation of a dexterity-type board game

Each of these aims will be evaluated by testing, but the techniques and tester groups required will vary to ensure all aims are covered.

Stability of the software will be evaluated by the developer, through a series of pre-written test cases. High-level cases are included in Appendix 7.2. These will be refined with more specific cases as each phase of development starts. In each internal testing cycle, all previous tests will be repeated to ensure previously working functionality has not been disturbed. It is unlikely that an automated testing regime will be possible due to the nature of the project and resources available. This internal testing has not been separately marked on the Gantt chart in Appendix 7.1, as this work will be conducted continuously throughout the development phase.

The other three aims detailed above can not be reliably evaluated by the developer alone. This will rely on recruiting volunteers from the developer's friends and family. As mentioned in the work plan section above, there will be several build points for the software, where testers will be able to evaluate the functionality ready at that point. This will form part of the ongoing development and bug-testing process. The later user testing periods will be more informative regarding evaluation against the project aims.

It will be important to recruit testers with a variety of previous experience, both with computer games, and with the board games on which the software will be based. The 'easy to pick up and play' criterion will be best evaluated by testers with limited experience of 3D computer gaming, and who have not been part of earlier testing rounds. The final criterion above will require testers with

some familiarity with the type of physical games serving as an inspiration for this project. All testers of the final product should be able to comment on the 'fun' criterion, however as this is the most subjective point it will be important to have a reasonable number of testers.

The testing methodology involving users will also change throughout the project. Initially users will only have a very limited build to test, which will require supervision from the developer to instruct the user what functionality is present, and what feedback is required. As the software is built out, less supervision should be required. For the final product, the software should be entirely self-explanatory (including user help where necessary), and effective feedback can only be achieved by unsupervised user testing.

The evaluation of the completed project will be based on interviews with the volunteer testers, following their unsupervised sessions with the software. Feeback will be sought on the key evaluation criteria: is the game fun to play? How easy is the game to play without experience? Does the game represent your experiences of playing similar games in reality?

6 Feature prototype

One of the key requirements for this project is the ability for the player to pick up and control a game piece with six degrees of freedom. The first prototype thus focuses on implementing a control scheme to achieve this.

The prototype was created within Unity; a simple scene consisting of a camera and light source, with a floor plane and several game pieces was created. The game pieces will eventually be constructed as more complex rendering and collision meshes, but for this prototype primitive shapes were used.

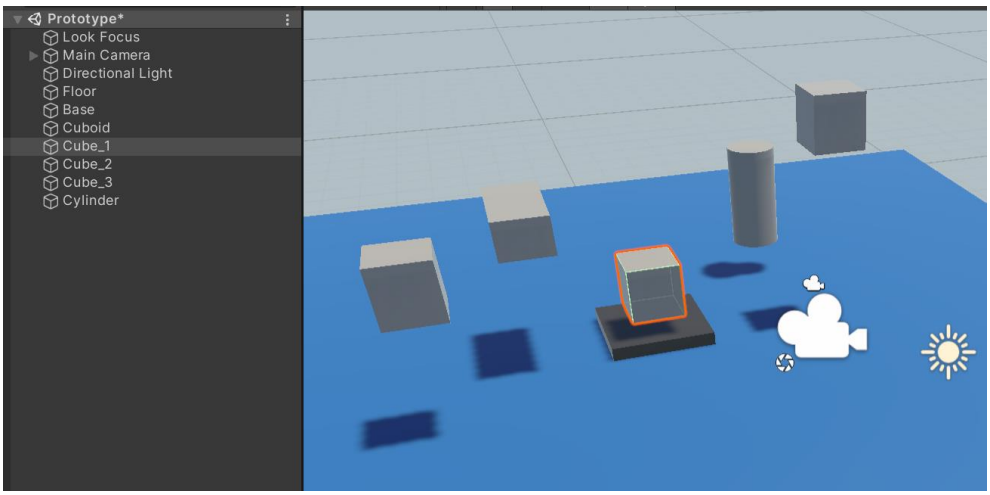


Figure 8: prototype scene

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 a C# script attached to the camera. The script is adapted from an example included one of the template projects bundled with Unity 2019.

A second script 'GrabObject' is also attached to the camera. This script allows the player to 'pick up' a game object, and to control it while it is in the 'held' state. When the player clicks the primary mouse button, a ray is cast through the cursor location. If an object tagged as a 'game piece' is hit, that piece is picked up. The object is childed to an empty transform (*holdFocus*) which is itself a child of the camera, and it is set to ignore gravity. While in this

state a game object will be held at a point in front of the camera, and thus be moved around the game world as the player moves the camera.

```
private void grabObject(GameObject target)
{
    //check target has an RB
    if (target.GetComponent<Rigidbody>())
    {
        //assign grabbed object
        heldObject = target;
        heldObjectRB = target.GetComponent<Rigidbody>();

        //set physics parameters
        heldObjectRB.useGravity = false;
        heldObjectRB.drag = 10;
        heldObjectRB.constraints = RigidbodyConstraints.FreezeRotation;

        //parent held object to grab area
        heldObjectRB.transform.parent = holdFocus;

        //set the object's tag
        heldObject.tag = heldTag;
    }
}
```

Figure 9: grabObject function within the GrabObject class

Attaching an object to the camera in this way allows it to be translated freely on the X and Z axes. There is a limited movement on the Y axis by rotating the camera, but to allow full movement the mouse scroll wheel is used. When scrolling while holding a game piece, the location of the '*holdFocus'* transform is moved with respect to the camera, moving the held object towards or away from the player.

Game piece rotation is achieved by activating a 'rotate' key, set as the space bar. While the space bar is held, the game switches into 'rotate' mode, in which the normal camera controls are disabled. This is achieved using the event system: the GrabObject class includes *onRotateStart* and *onRotateStop* events, to which the CameraController class subscribes and adds functions to disable and reenable camera controls. These events are invoked within the GrabObject class while the rotate command key is held. In this state, the GrabObject class instead interprets mouse XY movement and scrolling to rotate the held object around each of its axes.

```
private void rotateObject()
{
    //get mouse movement
    Vector3 mouseMovement = new Vector3(
        Input.GetAxis("Mouse Y") * -1,
        Input.GetAxis("Mouse X") * -1,
        Input.mouseScrollDelta.y * 5
        );

    heldObjectRB.transform.eulerAngles += mouseMovement;
}
```

Figure 10: rotateObject function within the GrabObject class

These two classes, along with the Rigidbodies and Colliders of the game objects, allow for full control of the game pieces with six degrees of freedom, which was the primary goal of this prototype.

Additional functionality important to the final product was also trialled. It will be important to know, for each game piece, whether it is successfully added to the player's growing structure, or if it falls off and is no longer counted. Each game piece starts with the 'unstacked' tag. A script is attached to each piece to detect collisions with other game objects. On a collision, the tag of the other object is read: if it is the structure base, or any piece tagged as 'stacked', the active piece is also tagged as 'stacked'. If a piece collides with the floor, it is tagged as 'grounded'.

At the same time, the game piece material is updated to give a visual indication of its status: grounded, held by the player, or stacked in the structure.

```
private void OnCollisionEnter(Collision collision)
{
    //do nothing while piece still held by player
    if (transform.gameObject.tag == heldTag)
    {
        return;
    }

    string otherTag = collision.gameObject.tag;

    //flag as grounded
    if (otherTag == floorTag)
    {
        transform.gameObject.tag = groundedTag;
        thisRenderer.material = unstackedMat;
        onStackChanged?.Invoke();
        return;
    }

    //flag as stacked if not grounded, and connected to base or existing stack
    if (
        (otherTag == baseTag || otherTag == stackedTag)
        && transform.gameObject.tag != groundedTag
        )
    {
        transform.gameObject.tag = stackedTag;
        thisRenderer.material = stackedMat;

        //broadcast event
        onStackChanged?.Invoke();
    }
}
```

Figure 11: updating a game piece tag and material on collision, within the PieceStacked class

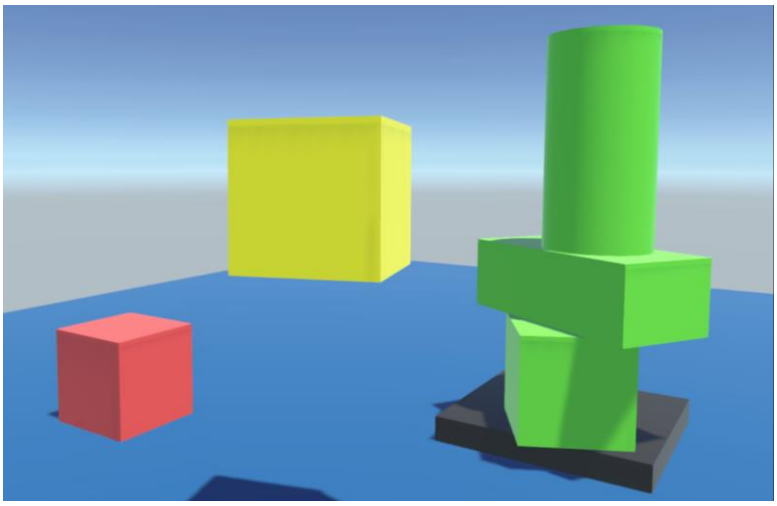


Figure 12: Game pieces in various states indicated by their material colour: grounded (red), held by player (yellow), stacked (green)

The final piece of functionality included in the prototype is the calculation of the total height of the player's structure. This will be necessary to determine which player is leading, and has won, a round.

A GameController script is attached to the camera. This class subscribes to the OnStackChanged event defined in the PieceStacked script attached to each game piece, and adds a CalcMaxHeight function to the event. Whenever the a piece is added to the player's structure (and tagged as stacked), or a piece touches the floor (having potentially fallen from the structure), the function loops through all stacked game pieces and finds the maximum Y value of the highest 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. The results are output to the debug console, but will be exposed to the players through the game interface.

```
private void calcMaxHeight()
{
    float maxHeight = 0f;

    GameObject[] stackedList = GameObject.FindGameObjectsWithTag(stackedTag);

    //loop through each object in the stack
    foreach(GameObject piece in stackedList)
    {
        //update max height with top of bounding box
        maxHeight = Mathf.Max(maxHeight, piece.GetComponent<Collider>().bounds.max.y);
    }

    Debug.Log("Stack height: " + maxHeight.ToString("0.0"));
}
```

Figure 13: calcMaxHeight function within the GameController class

The features implemented in the prototype serve to prove the feasibility of the first part of the project plan: a single-player implementation of the basic game mechanics. This will require building upon to flesh out the product into a viable game: custom game pieces with collision meshes, hot-seat multiplayer functionality, player displays, and a menu system.

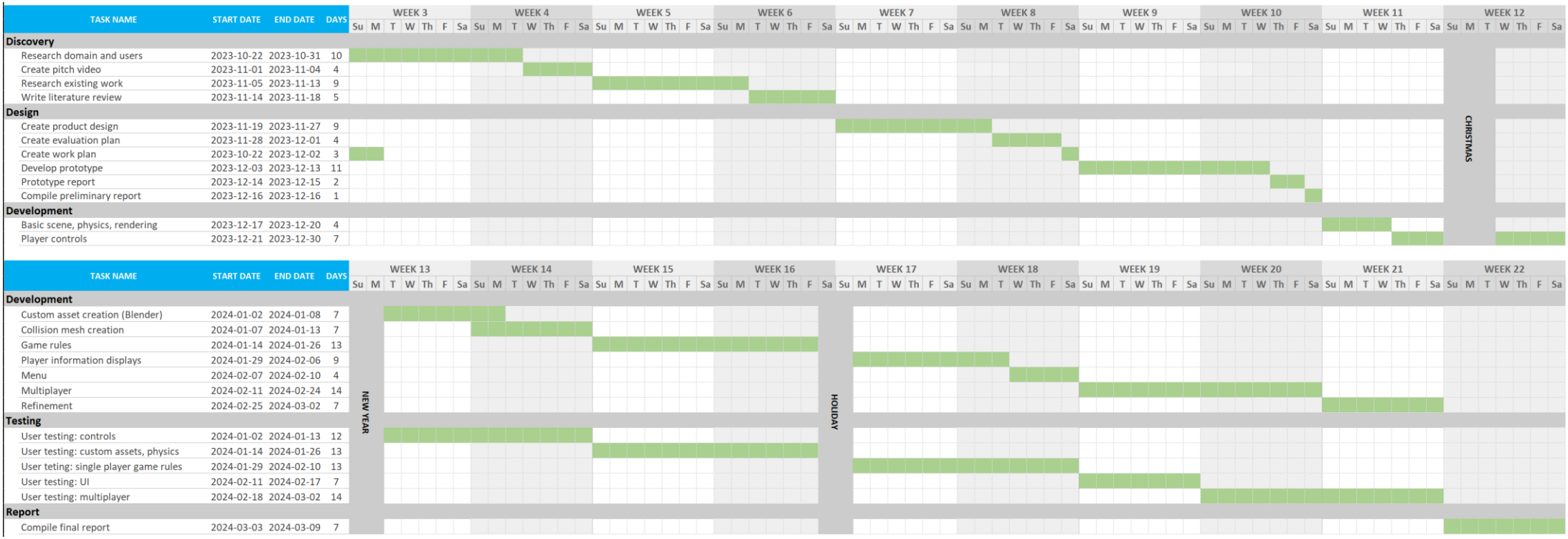
While the prototype does implement the features planned, it is not without issues. The camera moves very jerkily under the player controls, which makes precisely placing a held game piece difficult. Object rotation is not particularly intuitive; additional experimentation with this system is required. Further

research into how this problem has been tackled by other games should prove helpful with this. The interaction of the camera and game piece classes occasionally exposes intended behaviour, such as continued disabling of camera controls after a piece is dropped. Cases such as this will hopefully be caught by the playtesting regime planned throughout the development phase.

There has been limited opportunity for third-party testing of this prototype, but one other person has experimented with the software in its current state. This person is not a regular gamer, and has little familiarity with the WASD-and-mouse control scheme. Without instruction they were unable to effectively interact with the game world. This emphasises the need for, at the very least a menu listing the game controls. Ideally an interactive tutorial would be included in the final product, but this is likely to be beyond the scope of the resources available.

7 Appendices

7.1 Work plan



7.2 High-level test cases

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

Test Name	Main menu functions
Preconditions	Main menu displayed
Test Steps	Adjust player count Start game
Expected Results	Game starts with selected player count

Test Name	Camera movement
Preconditions	Game started
Test Steps	Use mouse and WASD keys to move camera around game world
Expected Results	Camera responds to input

Test Name	Game piece translation
Preconditions	Game started
Test Steps	Click and hold to pick up piece Use camera movement controls Use scroll wheel
Expected Results	Game piece is held at camera focus Game piece moves with camera Game piece moves to/from camera with scrolling

Test Name	Game piece rotation
Preconditions	Game piece held
Test Steps	Hold rotate button Move mouse, scroll wheel
Expected Results	Game piece rotates around three axes

Test Name	Gravity and collisions
Preconditions	Game piece held
Test Steps	Move game piece around world Release game piece
Expected Results	Held piece collides with other pieces, moving them Released piece falls under gravity, colliding with other pieces

Test Name	Add to structure
Preconditions	Game started
Test Steps	Use controls to add piece to player's structure
Expected Results	Piece detected as part of structure Structure height updated
Test Name	Structure collapse
Preconditions	Game started Player structure exists
Test Steps	Place piece such that at least one piece falls from structure
Expected Results	Fallen piece(s) detected Structure height updated Player score reduced

Test Name	Pass turn
Preconditions	Game started Active player has placed piece in their structure
Test Steps	Click 'pass turn' button
Expected Results	Next player camera set as active Active player indicator in interface updates

Test Name	Player wins
Preconditions	Game started
Test Steps	Active player reaches target structure height
Expected Results	Game end screen displayed Active player declared winner Scores displayed

Test Name	Return to menu
Preconditions	Game started
Test Steps	Click menu button
Expected Results	Main menu displayed Return to game in progress button displayed Start new game button displayed

Test Name	Return to game in progress
Preconditions	Game started Menu button clicked
Test Steps	Click return to game in progress
Expected Results	Game resumed in status as left

Test Name	Start new game
Preconditions	Game started Menu button clicked
Test Steps	Click start new game
Expected Results	New game starts with selected player count

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].