Can the learning mechanic – game mechanic framework be utilized to innovate serious game mechanics?

Commercial games development MSc

Games research and development

Research report

Samuel Scott

17019429

Submission date: 16/05/2023

Contents

Can the learning mechanic – game mechanic framework be utilized to innovate serious game mechanics?

Keywords

Learning mechanic – Game mechanic (LM-GM); Serious game mechanic (SGM); Pedagogical patterns; Game design patterns; Learning mechanic (LM); Game mechanic (GM)

Abstract

Serious games have become increasingly popular since their conception and have been utilized as an educational medium in a variety of industries. Despite this, there is a sparsity of tools available that can aid developers in identifying game mechanics and design patterns that could be used to convey a pedagogical pattern (Callaghan. *et al*, 2016). The Learning Mechanic – Game Mechanic (LM-GM) framework (Arnab. *et al*, 2014) is introduced as a model to help visualise the relationship between game patterns and learning patterns. The ongoing development of an environmental serious game, aimed at illustrating pykretes viability as a global warming solution, utilizes this LM-GM framework to identify potential game mechanics that can provide pedagogical outcomes, using the Unity game engine as a platform to innovate serious game mechanics for the serious game. This paper aims to contribute to the literature of serious game development, by providing a practical application of synthesising serious game mechanics based upon research that will aid in balancing pedagogical patterns and game design patterns.

Introduction

General game development has a plethora of literature that is available to developers, detailing game design principles and patterns. However, serious game development does not have the same quantity of information available that can aid developers in identifying how game design patterns can be utilized to convey pedagogical patterns. Therefore, by establishing game mechanics and pedagogical patterns that can be incorporated into the formulation of serious game mechanics (SGMs), could provide a foundation to innovate game systems that achieve the established learning outcomes (Tsai and Tsai, 2020; Hou, 2023).

This paper seeks to establish new systems and development processes, within Unity, in order to produce an environmental serious game as a component of the project and to illustrate the possible benefits of pykrete as a global warming solution. The environmental serious game that was developed in conjunction with the present paper employs insights from the review of academic literature to design new game systems for SGMs.

1. Literature Review

1.1. Literature introduction

Literature is comprised of the principles of serious game development, as well as serious games that have already been developed, or analysed, utilizing the learning mechanic – game mechanic framework established by Arnab *et al*. (2014). This framework provides a basis for mapping learning mechanics and game mechanics, resulting in serious game mechanics (SGMs). Further research consists of learning approaches that can aid in balancing pedagogical patterns with game design patterns.

1.2. What are serious games?

Serious games are unlike traditional video games, as they are used as an educational medium. Serious games, as known today, were first mentioned by Clark Abt, as defined by Abt (1970, cited by Dziak and PLECHAWSKA-WÓJCIK, 2017, p.15) serious games provide educational use, separate from entertainment. Since serious games were first concepted, the genre has seen worldwide growth within education, having an estimated market value of 5.84 million dollars in 2022. Furthermore, due to factors such as the corona virus pandemic leading governments into focusing on game-based learning, the industry has been forecasted to reach a market value of 32.72 billion dollars by 2030, as highlighted by Gaikwad (2022, cited in Tan and Nurul-Asna, 2023, p.20). This highlights how serious game are being recognised as a viable platform to achieve educational outcomes.

However, serious games are not limited to the education industry, they span a plethora of industries. As put forward by Alvarez and Djaouti, (2011), a serious game can be characterised by two points. Firstly, a serious game combines video game elements and various functions. These functions involve: broadcasting messages, providing training and systems that facilitate the exchange of data. Secondly, serious games are targeted to markets that are beyond the scope of the entertainment industry. These industries range from, but are not limited to, defence, training, education, commerce and communication.

While traditional video games can provide entertainment to a wide variety of audiences, serious games have been shown to typically serve a more targeted audience, which must be considered when developing a serious game. As well as meeting a specified target audience, a serious game must identify the functions required for achieving the required outcomes, established before the development of a serious game (Tsai and Tsai, 2020).

1.3. Learning Mechanic – Game Mechanic (LM-GM)

While serious games have aims that are beyond the scope of traditional video games, by providing purpose that is not solely entertainment based, serious games must still be engaging enough to allow the user a transference of learning through the game-mechanics provided within the serious game. While serious games have increased in popularity over the years, as put forward by Hou (2023), one of the biggest issues with educational serious games is their inability to integrate game design principles with educational principles. Therefore, the balance between fun and educational factors should be present from the planning stages of a serious game's development. Furthermore, identifying components that can provide the desired learning mechanics is a crucial factor in creating a serious game. Figure 1 defines the relationship between pedagogical patterns and game mechanics, thus providing a serious game mechanic (SGM). As stated by Arnab *et al* (2014), the complex relationships contained within a SGM are comprised of pedagogy, learning and

entertainment. Therefore, the formulation of a SGM requires the identification of both pedagogical patterns and game design patterns that relate to the desired educational outcomes of the serious game in development.

Figure 1. The relationships contained within a serious game mechanic (SGM) (Arnab. et al, 2014)

1.4. Formulating a Serious Game Mechanic (SGM)

To aid with serious game development the LM-GM framework was developed by Arnab. *et al* (2014). This framework has been created to help developers in both creating SGMs and identifying SGMs found while analysing an existing serious game. This framework has been utilized in the development of various serious games across different industries, such as Firewall and SQL Injection (Gaurav. *et al*, 2022) which are two web-based serious games developed in Unity, aimed at teaching the user about web security. The framework has also seen application as an analysis tool, being used to identify game mechanics and learning mechanics in existing serious games, such as Re-Mission (HopeLab, 2023) a renown serious game developed to provide engaging gameplay to young cancer patients, with the aim of highlighting the impacts and effects of successful cancer treatment. This serious game was analysed with the framework by Arnab. *et al*, (2014). The objective of this framework is to aid developers in mapping game mechanics to learning mechanics, providing a foundation for developers to balance pedagogical patterns with game design patterns. Figure 2 highlights several learning mechanics and game mechanics. While this list does not describe the concrete implementations of the resulting SGMs, it provides a basis to map game mechanics to learning mechanics, resulting in the construction of SGMs. The game mechanics and learning mechanics identified in figure 2 have been abstracted from literature relating to games studies and pedagogical theories.

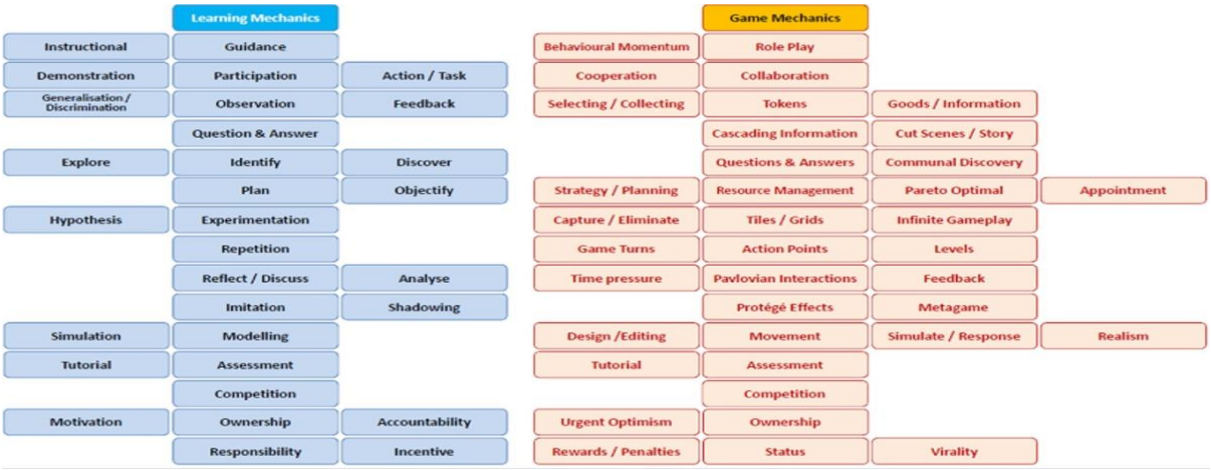


Figure 2. Examples of different learning mechanics and game mechanics (Arnab. et al, 2014)

Furthermore, Callaghan *et al*. (2016) proposed that the LM-GM framework allows for reflection on the pedagogical patterns and game elements that combine to create a serious game mechanic. This is illustrated in figure 3 and highlights how SGMs were identified for an existing serious game, aimed at providing a gaming medium for students to learn engineering principles. This highlights a practical use of the framework as an analytical tool to identify SGMs.

Table 4. Tone Generator extended LM-GM-based analysis

Game mechanic	Implementation	Learning mechanic	Description
Cut scene/Story	Pre-rendered videos explain the game objectives, mechanics and outcomes through storytelling	Instructional	Backstory sets game scenario. Player must fix the circuit to escape the room
Tutorials Cascading information	Tutorials at start of game guide user through basic mechanics of moving and biasing the circuits	Guidance/Tutorial	Player is guided through the initial stages of the game by informative graphics and cut scenes.
Simulate/Response	Player must select correct value of component(s) in circuit to achieve required output values/response.	Observation, Analyse Experimentation Modelling, Hypothesis	Game play tasks such as correctly biasing circuits provide the player with a sense of empowerment.
Movement	Navigate player token around 3D environment	Action/Task	Performing interactive tasks successfully and completing levels provides a sense of progress player satisfaction and game mastery.
Time pressure	Complete each level within time constraint		
Levels Feedback Assessment Meta-game	Advance to next level. Score shows time taken, accuracy and level of understanding of task completed	Feedback Motivation Assessment Reflect	End of level score reinforces sense of understanding and progress to maintain motivation. Provides benchmark for start of reflection process
Competition Rewards	Game leader board and achievements	Competition Motivation Incentive	Public leader board/achievement allows student to compare their score/performance to other players and try to improve.
Behavioral Momentum	Game play repeats itself through multiple levels to cause a shift in behaviour on the part of the player.	Repetition	Repetitive gameplay reinforces behaviour change. Player trials multiple strategies on the levels to increase overall performance.

Figure 3. Serious game mechanics identified for an engineering serious game (Callaghan. et al, 2016). Mechanics were identified utilising the LM-GM framework (Arnab. et al, 2014)

While figure 3 shows the mapping of LMs and GMs into the formulation of various SGMs, it does not indicate the approach utilized to balance pedagogical patterns with game design patterns. Despite this, figure 3 indicates how the LM-GM framework (Arnab *et al*., 2014) can be applied to a serious game to break down game elements into learning mechanics and game mechanics, which combine into a SGM. A game mechanic identified in figure 3 is cut scene and story, being mapped to an instructional learning mechanic. Although this shows a high-level overview of the relationship between the game mechanic and learning mechanic, it could provide more insight into the overall pedagogical approach used to identify the SGM. Lim *et al*. (2014) highlighted the use of a narrative based approach to learning, which could be used to describe the pedagogical approach used for the cutscene and story SGM shown in figure 3. Narratives can enhance the level of engagement by clarifying content. Furthermore, a narrative approach to learning can promote critical thinking, while increasing knowledge acquisition (Lim *et al*., 2014).

2. Methodology

2.1. Pykrete production simulation: Pykrete as a solution to global warming introduction

The serious game developed alongside this paper is a client-based project, aimed at highlighting the impact of producing pykrete bergs to reduce the effects of global warming. Pykrete is an impact resistant ice composite, created by freezing water with wood-based material (Firdaus. *et al*, 2020). The client purposed that pykrete ice bergs are created through harvesting trees and transporting lumber to the Antarctic peninsula, where the bergs are produced. The aim of producing these bergs is to capture carbon out of the atmosphere and reduce the effects of global warming. While the client has put forward the idea that pykrete could be a cost-effective viable solution to climate change, it is a theoretical science.

While pykrete as a global warming solution is theoretical, there is research to suggest that it could be utilized to absorb carbon from the atmosphere. As stated by Toochi (2018), one way of reducing the emissions of carbon dioxide into the atmosphere is to store CO2 in a separate medium than the atmosphere. Furthermore, the process of photosynthesis removes CO2 from the atmosphere and transfers it into organic carbon. However, when the process respiration occurs the natural carbon is converted back into CO2. Therefore, the pykrete solution proposes to process pulverised wood that has absorbed carbon and freeze it into bergs to reduce the rate at which the respiration process occurs, keeping more CO2 out of the atmosphere. Creating an effective solution to climate change is impactful as it has been suggested that the emissions generated by developed countries may need to be reduced by at least 80% by 2050 to mitigate the risk of climate-based disasters (Workman. *et al*, 2011), which may not be feasible.

Therefore, the target audience for this serious game is the United Nations, with the game's goal being to indicate to the UN's climate body that pykrete could be a potential solution to climate change and to potentially earn UN funding for the client to implement the theory.

Overall, the key outcomes requested by the client in the development of this serious game is to illustrate the pykrete production pipeline and highlight the cost effectiveness of pykrete as a global warming solution. As the concept of the serious game is to involve global participation, game mechanics should allow the user to place different tree types in different nations, as well as place different infrastructure and logistics, managed through an economy system. This aims to highlight the different components required to create a pykrete production pipeline and illustrate the costs associated.

Figure 4. In game screen shot of Pykrete bergs to Antarctica, the serious game developed alongside this paper

2.2: Identifying serious game mechanics through the LM-GM framework

As discussed in section 1.3 the LM-GM framework (Arnab *et al*., 2014) can be applied to analyse serious games, breaking down game elements into serious game mechanics (SGMs). These SGMs are comprised of game mechanics and learning mechanics. This framework will be utilized to identify game mechanics and learning mechanics found within the serious game developed alongside this paper. As the serious game developed alongside this paper was a team project, the SGMs identified within this serious game will contain systems developed by the whole team, shown in figure 5. However, as implementation of SGMs for this paper will involve disclosure of implementation methods, requiring an in depth understanding of the developed game systems, methodologies of SGM implementations will only include systems developed by the author of this paper.

Figure 5 highlights the SGMs that have been formulated for this environmental serious game, this has been achieved by mapping pedagogical patterns to game mechanics, utilizing the LM-GM framework as a foundation for visualizing the complex relationship between learning mechanics and game mechanics. The result of mapping a game mechanic to a learning mechanic is a SGM, one for each row, consisting of the game mechanic, learning mechanic and a brief implementation method and description of the SGM.

Game mechanic	*Implementation*	***Learning mechanic***	***Description***
Tutorials and cascading information	Introductory scenes providing information on background, game mechanics and objectives.	Guidance / tutorial	Scene describing game background and game objectives. Interactive tutorial scene providing information on different gameplay elements.
Simulate /response	Players must purchase and place infrastructure and logistics, as well as tree projects.	Participation	Ability to purchase and spawn different tree types. Ability to purchase and spawn different types of infrastructure and logistics.
Strategy / planning	Players must utilize tooltip data and map modes to plan where to grow trees, build infrastructure and place logistics.	Analyse	Various map modes players can analyse to inform different gameplay decisions. Tooltip data to inform players of different object data and nation data, such as tree projects optimum growth temperature and a nations average heat temperature.
Resource management	Numerous data and resources player must manage to achieve the games objectives.	Analyse	Various player (global) data the player can analyse to inform gameplay decisions, such as pykrete stockpile to produce pykrete bergs with tera factories. Nation and tile data the player can analyse to inform gameplay decisions.
Time pressure	Player must reverse climate change by producing enough pykrete bergs to reverse global warming.	Action/Task	Time system Monthly heat rise Player can produce pykrete bergs through tera factories.
Feedback	Visualisation of when player is achieving game objectives, as well as interacting with different game elements.	Motivation / Incentive	VFXs for building, planting trees, harvesting trees, and upgrading infrastructure. Nav mesh system, where producing bergs visualises them by moving them across antarctica.
Status / observation	Multipurpose graph providing the player with insight into objective based data across the entire playthrough.	Reflect	Bar chart and line chart Graph functions Various data sets the player can view through the graph
Assessment	JSON save system for saving various game data throughout the playthrough. Files can be viewed by the client and are easily readable	Reflect	JSON save system for player data, nation data, tile data and time data. Files are individually saved and can be accessed by the client for analysis
Selecting	The player can harvest trees, upgrade infrastructure and produce pykrete bergs	Repetition	Tree harvesting Infrastructure upgrade Pykrete berg production Nation and object data panels
Movement	Player can navigate a world map and manage tree projects, infrastructure and logistics	Action / Task	Camera movement system with clamping mechanics
Ownership	Visual effects to indicate infrastructure and logistics upgrades, as well as visual effects for players managing tree projects	Motivation / Incentive	Upgrade visual effects Infrastructure and logistics upgrade visual effects Tree planting visual effects Tree harvesting visual effects

Figure 5. Learning mechanics and game mechanics mapped into SGMs for the serious game developed alongside this paper

2.2.1. Balancing learning mechanics and game mechanics

With the main objectives of the development of this serious game being to highlight the potential use of pykrete as a global warming solution, there are many pedagogical patterns that could be utilized to inform users on the different components required to produce pykrete bergs. However, as pykrete as a global warming solution is theoretical, it will be unfamiliar to the user. Therefore, the educational process of learning by doing was utilized as one of the mediums to identify pedagogical methods that can facilitate a learning by doing approach to achieve the established outcomes in the development of this environmental serious game. As stated by Tan and Nurul-Asna (2023), environmental issues will typically require hands on actions and interventions. Therefore, providing users of this environmental serious game the ability to experience the process of creating pykrete ice bergs, as well as strategizing to manage resources required for the process, could enhance the knowledge acquisition within this serious game. Furthermore, as illustrated in figure 5, the tutorial mechanic highlighted utilizes a narrative approach to learning (Lim *et al*., 2014), as it sets the narrative that the player is part of the UN environmental panel, attempting to tackle climate change.

Figure 5 indicates the SGMs devised for this serious game, through applying the LM-GM framework (Arnab *et al*., 2014) and focusing primarily on providing a learning by doing approach to balancing pedagogical patterns with game design patterns. The SGMs formulated aim to allow the user to engage in the whole pykrete production process, facilitating the ability to strategize through resource management and providing the user with the ability to reflect and be assessed on their engagement within the serious game.

3. Serious game mechanic development methodologies

3.1. Participation: Object placement System

3.1.1 Pipeline for object asset creation and UI for respective objects

After discussion with the client, the idea of having various types of a certain object could be an engaging way for users to think about the production of pykrete. For example, redwood, although a large tree, takes longer to grow than a willow tree, so even though the yield of a redwood tree may be greater than that of a willow tree. The length it takes the red wood to grow would make it less effective of a tree when being used within the pykrete production process. Furthermore, climate influences the growth rate of trees and a willow being grown in the UK would fare better than a willow being grown within a desert. From discussion with the client, being able to allow the player to grow a redwood, or to grow a tree in a desert, is something they would like to see within the project. Due to this, creating an efficient pipeline for asset creation, which allows for easy implementation with game systems, is an important factor in implementing these requirements.

An asset pipeline has been implemented into game systems using scriptable objects. A scriptable object is a data container that is utilized to optimize memory usage within the Unity engine, while being independent of class instances. Utilizing scriptable objects allows the programmer to abstract any technical systems from the designer, while providing easy access to variables that are fundamental for designers.

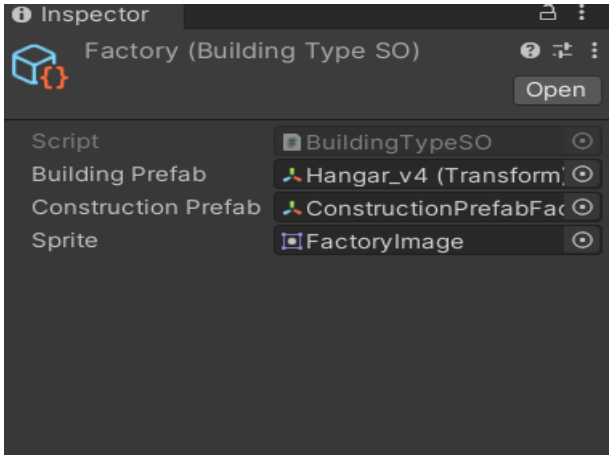


Figure 6. Scriptable object used to contain object placement variables

As shown in figure 6, the building prefab is the prefab of the actual object, for example a lumbermill, this is not instantiated straight away, first the construction prefab is instantiated and in a lower-level script has a variable for build duration. After the build duration has finished, the building prefab is instantiated, and the construction prefab is deleted from the scene. The sprite variable in the scriptable object is used for the image on the button for the respective object, as seen in figure 6. Inspiration was taken from the prototype pattern for respective UI buttons for object placement, where there is just one base button and new buttons are instantiated at run time, based off scriptable object variables.

```
Transform objectButtonTemplate = transform.Find("ObjectButtonTemplate");
objectButtonTemplate.gameObject.SetActive(false);

objectBtnDictionary = new Dictionary<BuildingTypeSO, Transform>();

int index = 0;

foreach(BuildingTypeSO buildingTypeSO in buildingTypeSOList)
{
    Transform objectButtonTransform = Instantiate(objectButtonTemplate, transform);
    objectButtonTransform.gameObject.SetActive(true);

    objectButtonTransform.GetComponent<RectTransform>().anchoredPosition += new Vector2(0, index * -60);

    objectButtonTransform.Find("BuildingImage").GetComponent<Image>().sprite = buildingTypeSO.sprite;
    objectButtonTransform.GetComponent<Button>().onClick.AddListener(() =>
    {
        buildingManager.SetActiveBuildingType(buildingTypeSO);
        UpdateSelectedVisual();
    });
    objectBtnDictionary[buildingTypeSO] = objectButtonTransform;
    index++;
}
```

Figure 7. Instantiating UI elements based off amount of building type scriptable object list and creating buttons through a dictionary

This implementation decouples button instantiation from the designer, providing a simple and comprehensible way for designers to add their object placement assets to the project, while also creating a respective UI element utilizing the dictionary, illustrated in figure 7.

3.1.2. Handling object placement validation, with Unity's physics overlap sphere

To handle object placement validation the Unity physics overlap sphere was used, in conjunction with a collider array. As it is undesired for objects to be placed on top of one another, or in water, validation was required to be able to define where an object should be able to be placed. Due to object placement being handled by a ray cast from mouse location, it made sense to check for a valid area to place an object within a defined radius. By utilizing Unity's physics overlap sphere this was possible, as it “computes and stores colliders touching or inside the sphere” (Unity Technologies, 2021). To use this sphere, the player raycast hit location is set as the center of the sphere, while a defined radius is set to define in what area objects of a certain tag are stored within the collision array. Using this system means unwanted objects such as water, or buildings can be set with their defined tag, this tag can be checked for as a Boolean, to check if it is within this collision array, and if so the object cannot be instantiated, as seen in figure 8.

```
private bool CanSpawnBuilding(BuildingTypeSO buildingTypeSO, Vector3 position)
{

    BoxCollider buildingBoxCollider = buildingTypeSO.constructionPrefab.GetComponent<BoxCollider>();

    bool isAreaClear = Physics.OverlapBox(position, buildingBoxCollider.size, Quaternion.identity) != null;


    if (!isAreaClear) return false;


    //Building of type within radius: this is used to check if there is a building of tag Building in a certain radius,
    // will be useful for putting lumber mills near trees for example

    float maxBuildingRadius = 15f;

    Collider[] colliderArray = Physics.OverlapSphere(position, maxBuildingRadius);

    foreach (Collider collider in colliderArray)
    {
        //Create bool variable of a tag you want to check for in collision arrray
        bool hasBuilding = collider.tag == "Building";
        bool hasWater = collider.tag == "Water";

        //Check for collision
        if (hasBuilding || hasWater)
        {
            //Could trigger UI here that indicates you can't build

            return true;
        }
    }

    return false;

}
```

Figure 8. Detecting if there is an object tagged with "Building" or "Water" in a defined radius

The prototype design pattern allows for the instantiation of similar objects, while being able to give each instantiation differentiating traits “the key idea is that an object can spawn other objects similar to itself. If you have one ghost, you can make more ghosts from it” (Nystrom, 2014).

The prototype design pattern is implemented by creating a new operation that is invoked by an object that represents that type. The state is stored on the instance itself, while behavior goes through a level of indirection and is delegated to the prototype and is stored on a separate object.

While using scriptable objects within this implementation means that the implementation is different from a typical prototype design pattern implementation, it is very much inspired by this design pattern. By being able to instantiate different buttons and objects based off scriptable object variables, using a singular button as a 'prototype', reduces the amount of repetitive code that would be involved manually programming functionality for each object placement button.

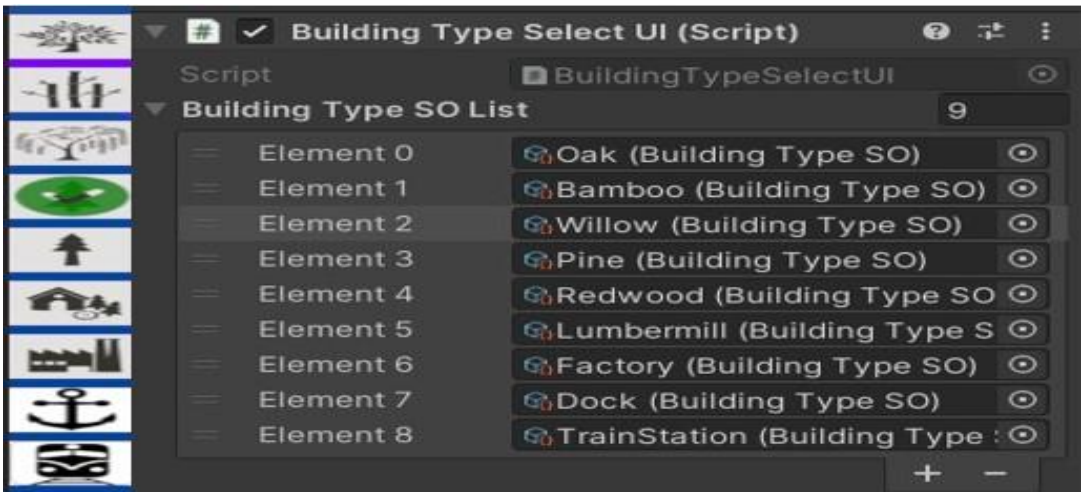


Figure 9. Building Type Select: responsible for instantiating buttons based off scriptable object sprites found within the Building Type scriptable object

3.2. Assessment / Reflect: Save and Load System

3.2.1 Save system: utilizing JSON for data serialization.

Data is a large component of this serious game and there are a variety of variables and data sets contained within the game. Therefore, developing a system that allows for this data to be saved to file is not only crucial for providing a save and load mechanism for the game, but also in providing the user a means to analyse different data sets after a finished playthrough. To achieve this java script object notation (JSON) was utilized to serialize game data into files and deserialize data from files. As stated by Smith (2014), JSON is derived from JavaScript and therefore only contains functions and data types found within JavaScript. However, JSON is not a programming language but a data interchange format. Therefore, JSON is utilized for saving game data to file, as it can be used to serialize varying data types and generate files that show the variables and their corresponding values, JSON files can also be put into website formatters that will further increase the legibility of JSON files.

```
public static void SavePlayerJSON(SavePlayerData obj, string fileName)
{
    string dir = Application.persistentDataPath + directory;
    string file = fileName + fileExt;

    if(Directory.Exists(dir))
    {
        string json = JsonUtility.ToJson(obj);
        File.WriteAllText(dir + file, json);
    }
}

public static SavePlayerData LoadPlayerData(string fileName)
{
    string file = fileName + fileExt;
    string fullPath = Application.persistentDataPath + directory + file;
    SavePlayerData so = new SavePlayerData();

    if(File.Exists(fullPath))
    {
        string json = File.ReadAllText(fullPath);
        so = JsonUtility.FromJson<SavePlayerData>(json);
    }

    else
    {
        Debug.Log("Save file does not exist");
    }

    return so;
}
```

Figure 10. Functions within JSON Manager script that handles saving and loading player data, utilising JSON

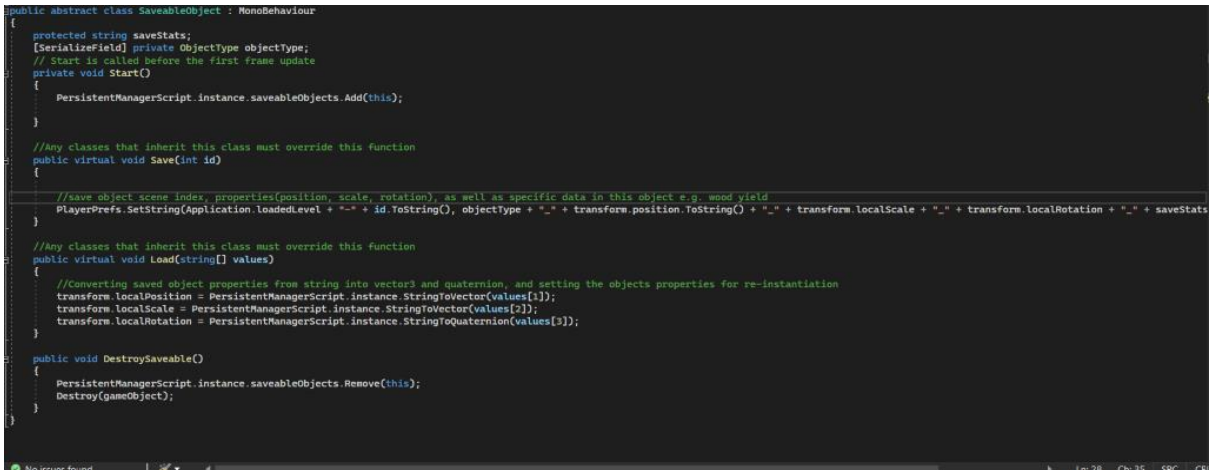
The JSON save system was implemented by first developing a JSON manager class, this class is a static class and therefore does not inherit from Unitys monobehaviour. This class is responsible for creating a directory in the user's persistent data path, the class also implements the means to serialize a data object into JSON and save it to file within the directory made within the persistent data path, as highlighted in figure 10. The JSON manager class also provides the means to deserialize a JSON file into an object that contains the variables found with the JSON file. This JSON system is utilized to serialize player data, nation data, tile data and time data. To be able to serialize each data set a data container needs to be implemented for each data set, as shown in figure 11. This container represents the object that is passed to its respective save function within the JSON manager script, which serializes the variables contained within the object into a JSON file. When loading these JSON files, each JSON file is deserialized into its respective data container, loading each variable from the JSON file into the container's variable, these containers are then used to populate local variables found within the game.

```
[System.Serializable]
public class SavePlayerData
{
    public int money;
    public int politicalPower;
    public int timber;
    public int pykrete;
    public int ships;
    public int trains;
    public float monthlyHeatLevelIncrease;
    public int trasportatedTimberWaitingToBeProcessed;

}
```

Figure 11. Data container for player data JSON file

3.2.2 Save System: utilising Unity player preferences and polymorphism for object serialization

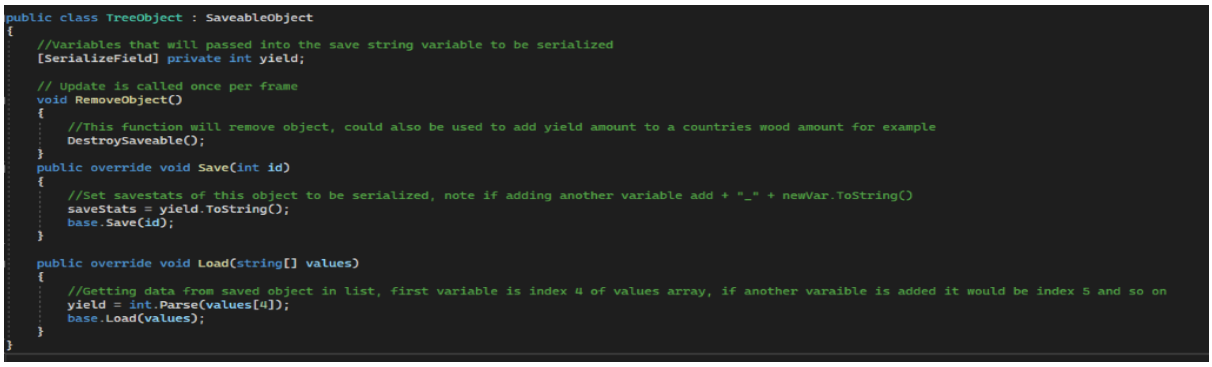
As object placement is a large element of player interaction in the game, such as placing trees or pykrete factories, being able to save and load objects would be a big factor while developing a save and load system for the game. However, as Unity's player preferences cannot store data types such as Vector 3's and quaternions, which is required for re-instantiating objects, there would need to be conversions from these values into strings to be able to serialize them. As these conversions would be required for every object type, with only changes in primitive variables, the object orientated technique polymorphism was utilized, removing the need to repeat code.



```
public abstract class SaveableObject : MonoBehaviour
{
    protected string saveStats;
    [SerializeField] private ObjectType objectType;
    // Start is called before the first frame update
    private void Start()
    {
        PersistentManagerScript.instance.saveableObjects.Add(this);

    }

    //Any classes that inherit this class must override this function
    public virtual void Save(int id)
    {
        //save object scene index, properties(position, scale, rotation), as well as specific data in this object e.g. wood yield
        PlayerPrefs.SetString(Application.loadedLevel + "-" + id.ToString(), objectType + "_" + transform.position.ToString() + "_" + transform.localScale + "_" + transform.localRotation + "_" + saveStats
    }

    //Any classes that inherit this class must override this function
    public virtual void Load(string[] values)
    {
        //Converting saved object properties from string into vector3 and quaternion, and setting the objects properties for re-instantiation
        transform.localPosition = PersistentManagerScript.instance.StringToVector(values[1]);
        transform.localScale = PersistentManagerScript.instance.StringToVector(values[2]);
        transform.localRotation = PersistentManagerScript.instance.StringToQuaternion(values[3]);
    }

    public void DestroySaveable()
    {
        PersistentManagerScript.instance.saveableObjects.Remove(this);
        Destroy(gameObject);
    }
}
```

Figure 12. Implementation of base abstract class that objects will inherit to define unique data to serialize

To avoid the base SaveableObject class being added as an object component within Unity, the class is marked as abstract “abstract classes have the option to declare that they implement an interface without needing to provide a full implementation” (Griffiths, 2020). This means that any object requiring save functionality will require its own class to be implemented that inherits from the SaveableObject base class. Within the base class the save and load function are marked as virtual, meaning any script inheriting from this base class will need to implement the save and load functions, overriding them with the override keyword “a virtual method is one that a derived type can replace” (Griffiths, 2020). With this implementation, it means any shared functionality can be defined in the base script once, while changes to what is saved and loaded can be added to each specific object that inherits from this base class.



```
public class TreeObject : SaveableObject
{
    //Variables that will passed into the save string variable to be serialized
    [SerializeField] private int yield;

    // Update is called once per frame
    void RemoveObject()
    {
        //This function will remove object, could also be used to add yield amount to a countries wood amount for example
        DestroySaveable();
    }
    public override void Save(int id)
    {
        //Set savestats of this object to be serialized, note if adding another variable add + "_" + newVar.ToString()
        saveStats = yield.ToString();
        base.Save(id);
    }

    public override void Load(string[] values)
    {
        //Getting data from saved object in list, first variable is index 4 of values array, if another varaible is added it would be index 5 and so on
        yield = int.Parse(values[4]);
        base.Load(values);
    }
}
```

Figure 13. Implementation of tree object saving that inherits functionality shown in figure 12, yet defines unique data to

These systems allow for object properties to be saved at an individual object level, then when they are spawned, they are added to an object list. This list is found in a singleton class, called PersistentDataScript. This script handles the serialization and deserialization of the contents of this list, as well as converting strings into vector 3's and quaternion's, allowing for the objects in the list to be re-instantiated.

Overall, with this system objects can have their properties saved and be loaded when a game session is reloaded obtaining the object files through the resources folder. These objects can save and load unique data at an individual object level, without the need to repeat functionality that is shared between placeable objects.

3.3. Analyse: Tooltip system, utilized to flag post the serious game

Due to the serious game having numerous elements and mechanics, a large amount of flag posting will be required to aid users into understanding different elements of the serious game. To achieve this a tooltip system was implemented, this tooltip is used to not only display user interface information, but in game information. The tool tip provides the user with information on nations, objects and user interface elements. In order to implement this system a tool tip manager class was implemented. This class is responsible for dynamically scaling a panel to the required size to fit the display the string provided to this class, while being able to be displayed at the players mouse position. This class is a singleton and is accessed through a static instance reference, meaning there can only be one instance of this script (Nystrom, 2014). The implementation is shown in figure 14.

```
public void ShowTip(string tip, Vector2 mousePos)
{
    tipText.text = tip;

    //Ternary operator to insure the pixel size is not above 200
    tipWindow.sizeDelta = new Vector2(tipText.preferredWidth > 200 ? 200 : tipText.preferredWidth, tipText.preferredHeight);

    tipWindow.gameObject.SetActive(true);
    tipWindow.transform.position = new Vector2(mousePos.x + tipWindow.sizeDelta.x * 2, mousePos.y);
}

protected void HideTip()
{
    tipText.text = default;
    tipWindow.gameObject.SetActive(false);
}
```

Figure 14. Singleton implemented for handling the instantiation of the tool tip, as well as the container panel scaling

To generate static strings utilizing the interface tool tip system the script must be placed on the UI object that the tool tip will display information for, providing the string that is displayed in the inspector for that UI element. The UI tooltip class inherits from IPointerEnterHandler and IPointerExitHandler, which allow for functionality to be triggered when the mouse hovers over a UI element and when the mouse exists that UI element. When the users mouse pointer hovers over a UI element with this tooltip script a co-routine will trigger, waiting a set amount of time before showing the respective tooltip by providing the string to the tooltip manager. When the users mouse pointer exits the UI element tool tip will close by triggering the close function in the tooltip manager, shown in figure 14. For adapting tooltip functionality this class can be inherited, overriding the show message function in the child implementation, removing the need to repeat unnecessary code. The tooltip system for in game object and nation data is identical, however the base implementation does not inherit from IPointerEnterHandler and IPointerExitHandlerand, but implements monobehaviours OnMouseOver and OnMouseExit functions.

```
public class HoverTip : MonoBehaviour, IPointerEnterHandler, IPointerExitHandler
{
    protected float timeToWait = 0.5f;
    protected string dataToShow;

    public void OnPointerEnter(PointerEventData eventData)
    {
        StopAllCoroutines();
        StartCoroutine(StartTimer());
    }
    public void OnPointerExit(PointerEventData eventData)
    {
        StopAllCoroutines();
        UIHoverManager.OnLoseFocus();
    }
    public virtual void ShowMessage()
    {
        UIHoverManager.OnMouseHover(tipToShow, Input.mousePosition);
    }
    protected IEnumerator StartTimer()
    {
        yield return new WaitForSeconds(timeToWait);
        ShowMessage();
    }
```

Figure 15 Implementation of base UI hover tip, differentiating implementations can inherit hover over and exit functionality

3.4 Status / Observation: Multipurpose graph

With data being a large component of the development of this serious game, providing users with a medium to view critical game data could provide a means to observe the impact of game choices, as well as providing a further method to interact with game data, separate from seeing user interface values changing. Therefore, the development of a graph to communicate critical data to the user is a game mechanic that can provide a foundation for the status / observation learning mechanic, identified during the development of this serious game.

The graph developed for this serious game takes the form of a multipurpose graph, comprised of two separate graph modes, being a line chart mode and bar chart mode. Furthermore, the graph has a number of functions that can allow the user to interact with the data sets, decreasing the amount of visible elements and increasing the amount of visual elements. For further sign posting the tooltip system was integrated into the graph, providing information on specific values in the graph such as data set names and functions provided. This information can be triggered by the user hovering their mouse over a graph button or value contained within the graph.

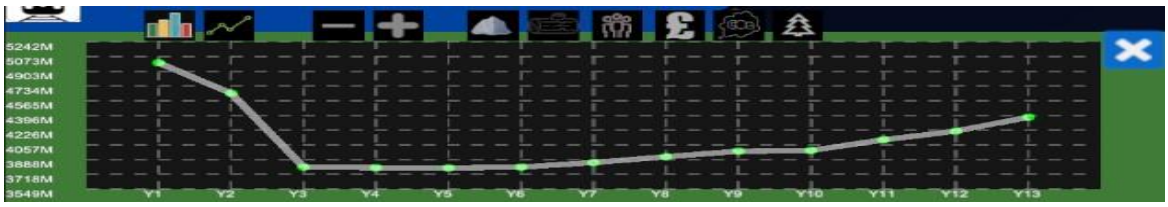


Figure 16. Line chart for global money data set after 13 in game years

Various C# principles were utilized in the development of this system, to increase not only the scalability of the system, but also the efficiency. Therefore, the graph can normalise axis values based on the data set provided and scale UI elements based upon the number of elements provided in the data set, while allowing for the addition of further data sets.

Figure 17. Line chart for global money data set after 13 in game years

Each element of the graph is comprised of a sprite, the line graph being formed from circle sprites and line sprites and the bar chat being comprised of square sprites, with their transform being scaled to represent their respective value. There is also a dash sprite which is tiled across the Y and X axis of the graph, this is to increase legibility of the graph, in-case the game is being played on a small display. The graph is contained in a parent object, this object is used to provide a persistent transform that can be used to scale graph elements depending on the amount of visible list elements that are being displayed from the data set. Functionality for the line graph and bar graph were abstracted into separate classes, contained within the same script, utilizing interfaces to dictate the methods that inherited classes implement. This means that data sets can easily be added to the graph through creating an instance of the desired graph type, shown in figure 18. The bar and line graph instantiation functionality are handled in their respective abstracted classes that implement the base interface.

```
//C02 Data graphs
IGraphVisual co2LineGraphVisual = new LineGraphVisual(graphContainer, dotSprite, Color.white, new Color(255, 0, 0, .3f));
IGraphVisual co2BarChartVisual = new BarChartVisual(graphContainer, Color.red, .8f);
```

Figure 18. Creating a line graph and bar graph for Co2 data set, created as an interface type with its value being the implementation of the interface for respctive graph type, providing required parameters to set up the graph

To further increase the maintainability of this graph system, a graph data manager was developed in the form of a singleton, used to maintain list values for each data set, such as adding values and interacting with the save system to save and load graph values, as well as providing new elements to the data set, that can then be viewed by the graph, as shown in figure 19.

```
ShowGraph(GraphDataManager.instance.co2ValueList, co2LineGraphVisual, -1, (int _i) => "Y" + (_i + 1), (float _f) => "°C" + _f);
```

Figure 19. Once a IGraphVisual graph type is created for a data set, as shown in figure 18, the graph type can be shown with the list being provided through the graph data manager singleton. Label values can also be set, with the X axis tag being set to "Y" for year and the Celsius symbol being show for the Y axis label.

4.Findings

How the project will be evaluated

Evaluation of this project will involve focus on two types of analysis, quantitative and qualitative. The quantitative aspect of the serious game's evaluation will involve a survey, which aims to gain an understanding of whether the serious game teaches users about pykrete as a possible climate change solution, as well as if the game is engaging to play. Following this, the second mode of analysis will involve qualitative analysis, focusing on a non-statistical evaluation of the project, such as client feedback, as well as reflection on the project and whether the LM-GM framework (Arnab *et al*., 2014) has been effective at creating game mechanics that balance pedagogical patterns with game design patterns.

Quantitative evaluation

For quantitative evaluation a prototype build was sent out to thirty people, along with a short questionnaire, to both game developer peers, and several non-game developer associates. In order to determine whether the serious game achieves its learning objectives, which include teaching people about the pykrete production chain. It is crucial to use a sample of non-experts and solicit their feedback on whether playing the serious game helped them understand the process (Saunders *et al*., 2020).

However, unfortunately only eight responses were obtained. While the survey was conducted (sample frame of 30 people) to establish how successful the serious game is at achieving its proposed learning outcomes, the sample frame was not of sufficient size (sample size of 8) to provide a definitive quantitative evaluation of the developed serious game (Henry, 1990 in Saunders *et al*., 2020, p.297).
Furthermore, as the prototype build was not available until later stages of the serious game's development cycle, the ethics form was not created and approved by the institution's ethical committee. Therefore, while anonymous, it would be academically irresponsible to disclose the data of the survey within the scope of this paper. Hence, an overview of survey findings will be used to indicate the initial engagement level of the serious game, as well as its effectiveness of achieving the required learning outcomes.

The survey consists of the following questions, with a symmetric Likert scale (Joshi *et al*., 2015) for each question.

- Were you aware of pykretes potential use as a climate change solution before engaging with this serious game? (1 - Not at all, 7 - Fully aware)
- To which extent would you say you understand the process required to produce pykrete bergs? (1 - Not at all, 7 - Fully)
- How engaging did you find the serious game? (1 - Not at all, 7 - Very much)

The main indication from the survey was that while the serious game was able to facilitate an environment where the user can learn about pykrete as a global warming solution, game mechanics present within the serious game could have been more engaging. Many researchers emphasised that fun is an effective way to promote the motivation of knowledge acquisition (Suttie *et al*., 2012; Tsai and Tsai, 2020; Hou, 2023). Therefore, by decreasing the amount of SGMs highlighted in figure 5 and focusing on a limited number of engaging mechanics, the pedagogical effectiveness of these mechanics could have been greater.

As the results of the survey are lacking, the wider applicability of the research needs further work (Saunders *et al.,* 2020). Therefore, as part of future research for this serious game, conducting a thorough survey, with completed ethics to allow for inclusivity within future development, could enhance in identifying areas for improvement.

Qualitative analysis

For the purpose of qualitative analysis, both peers and the client provided input. The client emphasised in the comments that the project had achieved the major objectives that were set forth before it was created, including exposing the Pykrete production pipeline and its potential for reducing global warming. The main criticism within feedback obtained from the client were mainly features that could be included with future development, such as the implementation of natural disaster effects, induced through global warming increases.

Feedback obtained from peers highlights how although the game has a variety of features and is large in scope, there are some game systems that could be more engaging to provide a higher level of user interaction (Hou, 2023). Through providing more user engagement the desired educational outcomes may be even more effective (Tan and Nurul-Asna, 2023), as well as making the serious game more enjoyable to play.

Summary of findings

Overall, the LM-GM framework has shown to be an effective tool, both in analysing serious games to identify SGMs, and in utilising the framework as a planning tool, to help visualise the complex relationship between pedagogical patterns and game mechanics. However, the framework does not guarantee that its application will allow developers to balance pedagogical patterns with game mechanics to achieve the required educational outcomes.

Through applying the LM-GM framework (Arnab *et al*., 2014), various SGM's were identified through mapping learning mechanics and game mechanics. These SGM's then had implementation methods devised for them. While the outcomes of this paper have determined that the framework can be utilized to innovate SGMs, it does not determine the level of engagement provided to the user from the developed SGMs.

Figure 5 highlights how there are many learning mechanics and game mechanics formulated for the development of SGMs for this serious game. While this provides various game mechanics for the user to interact with, by reducing the scope of the serious game and concentrating on the level of engagement for less mechanics, the overall user engagement could be improved.

Conclusion

While game development is a vast topic with an abundance of material addressing all aspects of game design, there is not nearly as much literature accessible to assist with the creation of serious games. As a result, providing developers with a way to visualize the intricate relationship between pedagogical patterns and game design patterns is critical in laying a foundation for the development of engaging serious game mechanics.

Within this paper the LM-GM framework has been established, devised by Arnab *et al*., (2014), and highlights previous applications of the framework. These findings are then utilized in the formulation of serious game mechanics developed for a serious game, created in conjunction with this paper. These game mechanics methodologies are then detailed, with evaluation of the project being composed of both quantitative and qualitative aspects.

Although this paper has identified that the established framework can be applied as both a planning tool and a development tool, it does not quantify the level of user engagement that SGMs created through the framework will provide. Therefore, further categorising the game mechanics and learning mechanics offered by the LM-GM framework (Arnab *et al*., 2014), which are highlighted in figure 2, could improve this framework by providing a way to balance pedagogical patterns with game mechanics to achieve a particular learning approach. Suttie *et al*. (2012) emphasised that a balance between education and entertainment is necessary for the development of an effective serious game. A learning approach utilized within the development of the environmental serious game, developed alongside this paper, is a learning by doing approach. As highlighted by Tan and Nurul-Asna (2023), facilitating a learning by doing approach to learning could enhance the knowledge acquisition process among users of the serious game.

Bibliography

Alvarez, J. and Djaouti, D. (2011) “An introduction to Serious game Definitions and concepts,” *Serious games & simulation for risks management* [online], 11(1), pp.11-15. [Accessed 15 January 2023].

Arnab, S., T, Lim., Carvalho, M., Bellotti, F., Freitas, S., Louchart, S., Suttie, N., Berta, R. and Gloria, A. (2014) “Mapping learning and game mechanics for serious games analysis,” *British Journal of Educational Technology* [online], 46(2), pp. [Accessed 16 December 2022].

Callaghan, M.J., McShane, N., Gomez Eguiluz, A., Teilles, T. and Raspail, P. (2016) “Practical application of the learning mechanics-game mechanics (LM-GM) framework for serious games analysis in Engineering Education,” *2016 13th International Conference on Remote Engineering and Virtual Instrumentation (REV)* [online], pp. 291–395. Madrid, Spain, 31 March 2016. IEEE. Available from: https://ieeexplore.ieee.org/document/7444510 [Accessed 20 January 2023].

Cecotti, H. and Callaghan, M.J. (2021) “Practical application of the learning mechanics-game mechanics framework for serious games design and analysis for the development of mental computation in virtual reality*,” 2021 IEEE International Conference on Engineering, Technology & Education (TALE)* [Preprint] [online]. Wuhan, China, 5-8 December 2021. IEEE. Available at: https://ieeexplore.ieee.org/document/9678639 [Accessed 10 January 2023].

DZIAK, L. and PLECHAWSKA-WÓJCIK, M. (2017) “The use of Unity 3D in a serious game dedicated to development of firearm handling skills,” *Applied Computer Science* [online], 13(2), pp. 15–22. [Accessed 16 December 2022].

Firdaus, P., Sulistijono, R. and Ardhyananta, H. (2020) “Study of pykrete viability as protective insulation in cold storage application,” *INTERNATIONAL CONFERENCE ON SCIENCE AND APPLIED SCIENCE (ICSAS2020)* [online], 2296(1), pp. 1–7. Surakarta, Indonesia, 7 July 2020. AIP Publishing. Available at: https://pubs.aip.org/aip/acp/article/2296/1/020082/724096/Study-of-Pykrete-viability-as-protective [Accessed 25 April 2023].

Gaurav, D., Kaushik, Y., Supraja, S., Yadav, M., Gupta, M. and Chaturvedi, M. (2022) “Empirical study of adaptive serious games in enhancing learning outcome,” *International Journal of* Serious *Games* [online], 9(2), pp. 27–42. [Accessed 14 February 2023].

Griffiths, I. (2020) “Inheritance,” in *Programming C# 8.0: Build cloud, web, and desktop applications* [online]. Beijing: O'Reilly, pp. 267–300. [Accessed 10 January 2023].

Haas, J. (2014) “A History of the Unity Game Engine,” *Worcester Polytechnic Institute: Interactive Qualifying Projects* [online], pp. 1–26. [Accessed 11 February 2023].

HopeLab (ed.) (2023) *Re-mission.* Available at: https://hopelab.org/case-study/re-mission/ [Accessed 15 February 2023].

Hou, H.-T. (2023) “Diverse development and future challenges of game-based learning and gamified teaching research”, *Education Sciences* [online], 13(4), pp. 337–339. [Accessed 20 February 2023].

Joshi, A., Kale, S., Chandel, S. and Kumar Pal, D. (2015) “Likert scale: Explored and explained”, *British Journal of Applied Science & Technology* [online], 7(4), pp. 396–403. [Accessed 2 May 2023].

Lim, T., Louchart, S., Suttie, N., Baalsrud Hauge, J., Stanescu, I., Ortiz, I., Moreno-Ger, P., Bellotti, F., Carvalho, M., Earp, J., Ott, M., Arnab, S. and Berta, R. (2014) "Narrative serious game mechanics (NSGM) – insights into the narrative-pedagogical mechanism", *4th International Conference on Serious Games, GameDays 2014: Games for Training, Education, Health and Sports* [online], 8395, pp. 23–34. Darmstadt, Germany, 1-5 April 2014. Springer Verlag. Available from: https://link.springer.com/chapter/10.1007/978-3-319-05972-3_4 [Accessed 15 April 2023].

Miljanovic, M.A. and Bradbury, J.S. (2020) "GIDGETML," *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* [online], pp. 184–192. Seoul, South Korea, 27 June – 19 July 2020. Association for Computing Machinery. Available from: https://dl.acm.org/doi/10.1145/3377814.3381716 [Accessed 19 March 2023].

Nystrom, R. (2014) "Prototype," in *Game Programming Patterns* [online]. États-Unis: Genever Benning, pp. 59–72. [Accessed 2 March 2023].

Nystrom, R. (2014) "Singleton," in *Game Programming Patterns* [online]. États-Unis: Genever Benning, pp. 73–86. [Accessed 2 March 2023].

Saunders, M., Lewis, P. and Thornhill, A. (2020) *Research methods for business students* [online]. Eighth edition. London: Pearson. [Accessed 2 May 2023].

Singh, S. and Kaur, A. (2022) "Game development using Unity Game Engine," *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)* [Preprint] [online]. Rajpura, India, 18-19 November 2022. IEEE. Available at: https://ieeexplore.ieee.org/document/10007155 [Accessed 20 January 2023].

Smith, B. (2015) "Introducing JSON," in *Beginning json* [online]. Berkeley, CA, California: Apress, pp. 37–38. [Accessed 20 March 2023].

Suttie, N., Louchart, S., Lim, T., Macvean, A., Westera, W., Brown, D. and Djaouti, D. (2012) "Introducing the "serious games mechanics" a theoretical framework to analyse relationships between "game" and "pedagogical aspects" of serious games", *Procedia Computer Science* [online], 15, pp. 314–315. [Accessed 23 March 2023].

Tan, C.K. and Nurul-Asna, H. (2023) "Serious games for environmental education," *Integrative Conservation* [online], 2(1), pp. 19–42. [Accessed 25 April 2023].

Toochi, E.C. (2018) "Carbon sequestration: How much can forestry sequester CO2?," *Forestry Research and Engineering: International Journal* [online], 2(3), pp. 148–150. [Accessed 20 April 2023].

Tsai, Y.-L. and Tsai, C.-C. (2020) "Review for "A meta-analysis of research on Digital Game-based Science Learning", *Journal of computer assisted learning* [online], 36(2), pp. 280–294. [Accessed 5 April 2023].

Unity Technologies. (2018) *ScriptableObject*. Unity Technologies. Available from: https://docs.unity3d.com/Manual/class-ScriptableObject.html [Accessed 15 December 2022].

Unity Technologies. (2021) *Physics.OverlapSphere*. Available from: https://docs.unity3d.com/ScriptReference/Physics.OverlapSphere.html [Accessed 3 January 2023].

Unity Technologies. (2021) *PlayerPrefs*. Available from: https://docs.unity3d.com/ScriptReference/PlayerPrefs.html [Accessed 27 December 2022].

Workman, M. McGlashan., Chalmers, H. and Shah, N. (2011) “An assessment of options for CO2 removal from the atmosphere”, *Energy Procedia* [online], 4, pp. 2877–2884.
[Accessed 25 April 2023].

Appendix

Technical analysis

As the serious game was developed within the Unity game engine, the built in profiler was utilized to gain an insight on the performance of the software. The profiler breaks down the game's performance into various components, such as scripts, rendering and garbage collection. Information about these different components are provided through the profiler, such as execution time in milliseconds, as well as memory usage. The profiler can be deployed in both the editor, as well as within a development build, allowing for the profiler to be used as a standalone application.

With performance being a crucial factor in how the success of a game is perceived, the profiler was used to inform optimization decisions throughout the development of this serious game. Through utilizing the profiler to not only identify memory issues with scripts, but also understand what elements were affecting performance, such as rendering, the overall performance of the project was improved iteratively. By having an iterative approach to optimisation, the resulting final build has a faster build time and more stable frame rate in comparison to early prototype builds. Therefore, for a conclusive analysis of quantitative aspects of analysing this serious game, a snapshot will be taken from a final build of the serious game.

This snapshot will be taken from later stages of the game, where there is a high quantity of populated objects, audio and visual effects. Figure 20 will be evaluated to provide a quantitative analysis of the final project. While the profiler can show graphical profiling, the author was not responsible for asset design and optimisation and therefore does not have an in depth understanding of the optimization technics utilized for asset design.

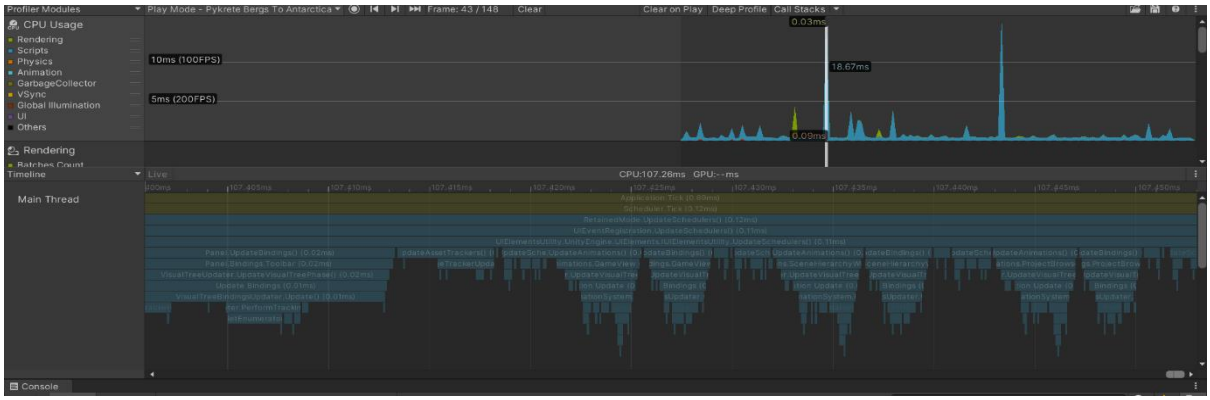


Figure 20. Snapshot of profiler detailing CPU performance when the game is populated with more objects than would typically be seen in a normal playthrough, by creating a build with large values for resource variables

Overall, as illustrated in figure 20, when the game is fully populated with all navigation Ai being executed the performance reaches a peak of around 70 frames per second. However, this amount of world population is not possible in a normal playthrough and was achieved by giving infinite money and pykrete, in order to push the performance limits of the build. As highlighted in figure 20, the highest latency found in a single execution is Application.Tick() and Scheduler.Tick(), while all the visible script executions are from Unity background scripts. This highlights how there are no scripts that have been developed during this serious game that are causing high latency, therefore the design patterns utilized during development have been successful in increasing performance of the software. However, an area for improvement is heap allocation, due to the nature of the game being the user places a large quantity of objects, these objects are allocated to the heap.

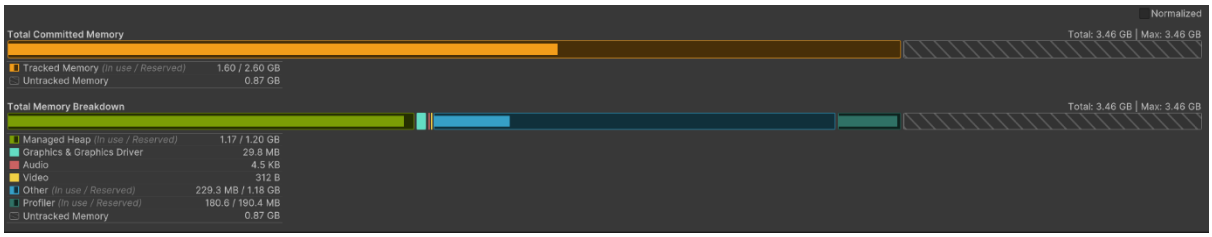


Figure 21. Breakdown of maximum memory usage throughout the game, obtained from the unity profiler

Through analysing memory usage, from the captured gameplay shown in figure 21, heap usage reaches a maximum usage of 1.20 gigabyte. This is due to the pooling system implemented for handling the repetitive instantiation and destruction of various objects. While partitioning memory for these objects increases heap allocation, this memory allocation is not required at runtime, therefore increasing CPU performance. However, as highlighted in figure 20 the frame rate can drop to around 70, from a normal rate of 200 during normal gameplay. This is due to the number of objects that are being rendered at once and improvements could be made utilising more asset creation optimisation techniques, such as level of detail, or billboarding.