Wydział Matematyki Stosowanej

Development of Computer Games

Farming game

Group, Name and Surname:
PI , Wojciech Kozieł
PI , Tatiana Cieślar
PI , Piotr Hadam
PAM , Dominika Limanówka

Date: 28 January

2021

Tables of Contents

Work Distribution Table:

Name/Surname	***Description of game development part***
Wojciech Kozieł	*Main character graphics and movement,* *Camera movement,* *Map design, Map animations,* *Temperature mechanism,* *Fire camp as object*
Tatiana Cieślar	*Toolbar and inventory management,* *Opening dialogue,* *Starter chest,* *Cutting down trees,* *Connecting toolbar to all mechanisms in the game (i.e. shop, chest, trees)*
Piotr Hadam	*Crop management,* *Mechanics of patches (i.e. digging, seeding, watering),* *Plant growth,* *Interactions with tiles,* *Marker, Main Menu, "About" Screen, Pause Screen*
Dominika Limanówka	*Interface and mechanism of the store (parrot) with animation,* *Daily addition of money,* *Sound manager - background music, sounds of individual elements and toggles,* *Sliders representing values of parameters of the character and mechanics of health and hunger loss,*

Description of Your Game

Description of Your Game.

1. 3D or 2D? *2D*
2. What type is your game? *Simulation*
3. What genre is your game? *Construction, management and survival simulation*
4. Platforms (mobile, PC or both?) *PC*
5. Scenario Description. *The player is shipwrecked on a desert island. Your goal is to survive - find tools, seeds and most importantly, food, also cut down trees to light a fire that will give you the warmth they need to survive at night. At the start of the game you need to find a chest with all of the necessary tools in it. You also need to grow crops to get enough food to eat. For every action the player needs to use a different tool: an axe to cut trees - with 5 strokes, a shovel to remove grass and weeds, a hoe to dig the ground and a watering can to water the crops. They can be collected in a bag. The player can grow corn, parsleys, potatoes, strawberries and tomatoes. You can get the seeds from the parrot sitting on a rock. Different seeds have different prices due to their growing time and health and hunger increase. Perform actions in the game to gather points and afford them. The goal of the game is to survive as many days as possible.*

GAME FEATURES

List of features (main functionality of our project)

1. Seeding plants
2. Plant growth
3. Toolbar and inventory mechanism
4. Shop mechanism
5. Survival mechanism
6. Night and campfire
7. Cutting down trees

Feature #1. *Seeding plants*

There is a mechanism in our game that is responsible for seeding plants and preparing for it. It recognizes what kind of base[1] is on the tilemap and, depending on this base and tool chosen, performs the appropriate action. It means you can remove grass, plow the ground and then, finally seed a plant. So we have 4 kinds of bases connected with seeding: grass, dirt, plowed dirt and seeded.

Figure 1. Kinds of bases

Some code samples showing how this mechanics works. Function SeedCrop works with different names of seeds, so it has different conditions but all are similar and we can show only one of them.

```
public void Mow(Vector3Int position)
{
    groundTilemap.SetTile(position, mowed);
}
```

Table 1. Removing grass and weeds

```
public void Plow(Vector3Int position)
{
    groundTilemap.SetTile(position, plowed);
}
```

Table 1. Plowing the ground

```
public void SeedCrop(Vector3Int position, string name)
{
    Crop cropSeeded;
    //depended on what plant we want to seed
    if (name == "corn")
    {
        cropSeeded = Instantiate(corn); //clone sample corn
        cropSeeded.position = position; //assign a position
        cropSeeded.state = cropSeeded.state0; //assign initial state
        cropSeeded.timeRemaining = 120; //assign time to grow

        crops.Add(position, cropSeeded); //add to dictionary of all crops
        corns.Add(position, cropSeeded); //add to dictionary of all corns
        cropTilemap.SetTile(cropSeeded.position, cropSeeded.state0); //change a tile on tilemap with crops
    }
    groundTilemap.SetTile(position, toWater); //change a tile on tilemap with ground
```

Table 1. Seeding plants

```
//if there is no plant on tile
if (crops[(Vector2Int)selectedTilePosition].noPlant)
{
    //usage of tools if tile has suitable ability
    if (fields[(Vector2Int)selectedTilePosition].ableToMow && toolbarController.GetItem.Name == "Shovel"
        && shopPanel.isOpen == false)
    {
        cropsManager.Mow(selectedTilePosition);
    }
    else if (fields[(Vector2Int)selectedTilePosition].plowable && toolbarController.GetItem.Name == "Hoe")
    {
        cropsManager.Plow(selectedTilePosition);
    }
    else if (fields[(Vector2Int)selectedTilePosition].ableToSeed && toolbarController.GetItem.isSeed == true)
    {
        switch (toolbarController.GetItem.Name) //depending on what seed you have chosen
        {
            case "Seeds_Corn":
                // Checking whether we have more than 4 seeds to seed
                if (GameManager.instance.inventoryContainer.slots[toolbarController.selectedTool].count
                    >= cornSeedsCount)
                {
                    cropsManager.SeedCrop(selectedTilePosition, "corn");
                    GameManager.instance.inventoryContainer.RemoveItem(toolbarController.GetItem, cornSeedsCount);   // Deletes 4 seeds
                }
            break;
```

Table 1. Usage of the above functions

Feature #2. *Plant growth*

Once you seed a plant, you have to water it to start its growth. All plants have their own timer and time required to achieve the next stage of growth, i.e. corn has 6 stages and needs 2 minutes to achieve one while parsley has only 5 [2] and needs a minute to achieve one. You have to water the plant every time it changes the stage. The last stage of each plant isn't waterable, you can only collect it.

Figure 2. Stages of parsley growth (the last one is ready to collect)

Code fragments responsible for watering crops and its growth will be presented below.

```
public void Water(Vector3Int position)
{
    crops[position].timerIsRunning = true; //init a timer - plant can grow
    groundTilemap.SetTile(position, watered); //change a tile on tilemap with gorund

}
```

Table 1. Watering plants - timer starts running

```
void Grow(Crop crop)
{
    if (crop.timerIsRunning) //if watered - timer is running
    {
        if (crop.timeRemaining > 0)
        {
            crop.timeRemaining -= Time.deltaTime; //counting down
            //Debug.Log(DisplayTime(crop.timeRemaining));
        }
        else
        {
            if (crop.name == "Parsley(Clone)") //parsley has less states of grow
            {
                //update state
                if (crop.state == crop.state0)
                    crop.state = crop.state1;
                else if (crop.state == crop.state1)
                    crop.state = crop.state2;
                else if (crop.state == crop.state2)
                    crop.state = crop.state3;
                else if (crop.state == crop.state3)
                    crop.state = crop.state4;

                cropTilemap.SetTile(crop.position, crop.state); //change a tile on tilemap with crops to next state
                groundTilemap.SetTile(crop.position, toWater); //change a tile on tilemap with ground to waterable


                crop.timerIsRunning = false; //timer stops counting
                if (crop.state != crop.state4)
                {
                    crop.timeRemaining = 1; //time to next state
                }
            }
            else
            {
                //update state
                if (crop.state == crop.state0)
                    crop.state = crop.state1;
                else if (crop.state == crop.state1)
                    crop.state = crop.state2;
                else if (crop.state == crop.state2)
                    crop.state = crop.state3;
                else if (crop.state == crop.state3)
                    crop.state = crop.state4;
                else if (crop.state == crop.state4)
                    crop.state = crop.state5;

                cropTilemap.SetTile(crop.position, crop.state); //change a tile on tilemap with crops to next state
                groundTilemap.SetTile(crop.position, toWater); //change a tile on tilemap with ground to waterable

                crop.timerIsRunning = false; //timer stops counting
                if (crop.state != crop.state5)
                {
                    if (crop.name == "Corn(Clone)")
                        crop.timeRemaining = 120; //time to next state
                    if (crop.name == "Potato(Clone)")
                        crop.timeRemaining = 90; //time to next state
                    if (crop.name == "Strawberry(Clone)")
                        crop.timeRemaining = 90; //time to next state
                    if (crop.name == "Tomato(Clone)")
                        crop.timeRemaining = 60; //time to next state
                }
            }

        }
    }
```

Table 1. Plant growth - if it's watered and timer is running timer is counting down, and if it reaches 0, stage of plant changes, it is executed in Update function

Feature #3. *Toolbar and inventory mechanism*

When it comes to the toolbar and inventory mechanism we decided to connect them. Instead of creating two very similar functionalities we use one as the primary mechanism (the inventory) [1] and the other as the derived one (the toolbar) [2]. We can exchange items in their slots by drag and dropping them - the first row of items in the inventory is present in the toolbar. The sprite of the dragged item is visible so that it's easier to shift them. Automatically the first item in the toolbar is highlighted, but we can highlight any item we want to. The inventory is connected to all the necessary elements of the game, e.g. trees, the bonfire and the shop (items are removed or added depending on the action the player is performing). Inventory can be opened by clicking E on the keyboard.

Figure 1. The inventory

Figure 2. The toolbar

All of the code below is responsible for the inventory mechanism, and therefore also for the toolbar. Code in Table 1 is placed in the ItemPanel.cs class which is the base class for two derived classes: ItemPanel (the inventory) and ItemToolbarPanel (the toolbar). They are responsible for both of the panels in the game. The other difference is the ToolbarController and the InventoryController which are slightly different in functionality. The rest of the classes for both of the functionalities is identical.

```
Odwołania: 2
public void Init()
{
    SetIndex();    // Cicles through inventory and sets indexes to the buttons
    Show();
}

1 odwołanie
private void SetIndex()
{
    for (int i = 0; i < inventory.slots.Count && i < buttons.Count; i++)
    {
        buttons[i].SetIndex(i);
    }
}

Odwołania: 3
public void Show()
{
    for (int i = 0; i < inventory.slots.Count && i < buttons.Count; i++)
    {
        if (inventory.slots[i].item == null)
        {
            // Hiding what the slot in inventory contains in the button
            buttons[i].Clean();
        }
        else
        {
            // Setting the button to the item in the inventory
            buttons[i].Set(inventory.slots[i]);
        }
    }
}
```

Table 1. Setting and displaying items in the inventory in the ItemPanel.cs class

```
Odwołania: 12
public void Add(Item item, int count = 1)
{
    // Determining if the item is stackable
    if (item.stackable)
    {
        // Finding slot with the same item
        ItemSlot itemSlot = slots.Find(x => x.item == item);


        if (itemSlot != null)
        {
            itemSlot.count += count;  // Adding the count
        }
        else
        {
            // If the item doesn't exist yet it is added
            itemSlot = slots.Find(x => x.item == null);

            if (itemSlot != null)
            {
                itemSlot.item = item;
                itemSlot.count = count;  // 1 item
            }
        }
    }
    else
    {
        ItemSlot itemSlot = slots.Find(x => x.item == null);

        if (itemSlot != null)
        {
            itemSlot.item = item;
            itemSlot.count = count;
        }
    }
}
```

Table 1. Adding items to the inventory

```
1 odwołanie
internal void OnClick(ItemSlot itemSlot)
{
    // Puts the item from inventory to the drag and drop slot
    if (this.itemSlot.item == null)
    {
        this.itemSlot.Copy(itemSlot);
        itemSlot.Clear();
    }
    else
    {
        // If the item slot is not empty we exchange items inside the item slots

        Item item = itemSlot.item;
        int count = itemSlot.count;

        // Assigns the currently dragged item into the inventory item slot
        itemSlot.Copy(this.itemSlot);

        this.itemSlot.Set(item, count);
    }
    UpdateIcon();
}
```

Table 1. Dragging and dropping elements in the inventory

```
Unity Message | Odwołania: 0
void Update()
{
    if (Time.timeScale == 0)
        return;

    // Opens/closes inventory after clicking E (equipment) on keyboard
    if (Input.GetKeyDown(KeyCode.E) && !dialogue.activeSelf)
    {
        panel.SetActive(!panel.activeInHierarchy);
        toolbarPanel.SetActive(!toolbarPanel.activeInHierarchy);
        if (panel.activeInHierarchy)
            isOpen = true;
        else
            isOpen = false;
    }
}
```

Table 1. Opening or closing inventory after clicking E on keyboard

Feature #4. *Shop mechanism*

The character on the island is quite lonely and abandoned, and the only companion character is a green parrot. The bird is not able to help him very much in his survival, but in exchange for golden coins, it provides him with different kinds of seeds. To do business with a parrot, simply approach it. It always greets the character with a cheerful squeak.

Figure 1. Parrot

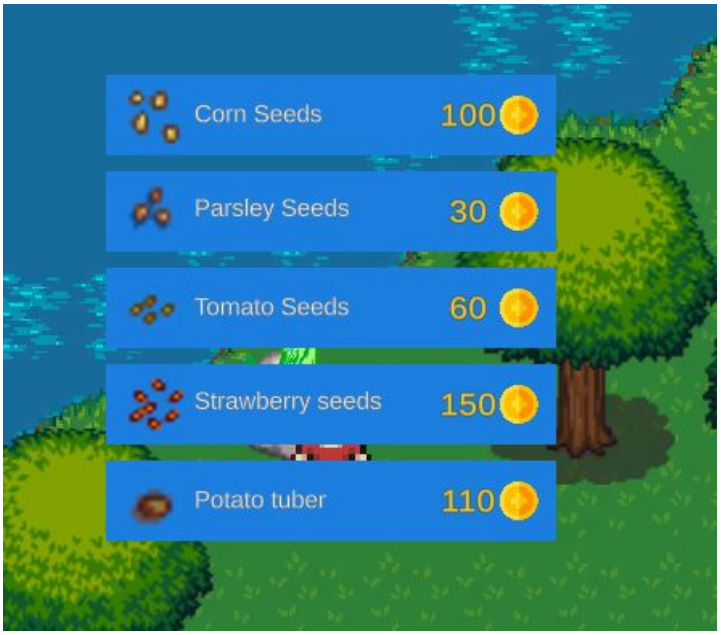

Figure 2. Shop UI

```
public class ShopTriggerController : MonoBehaviour
{
    [SerializeField] private UI_ShopController uiShop;


    private void OnTriggerEnter2D(Collider2D collider)
    {
        uiShop.Show();
        FindObjectOfType<SoundManager>().Play("Parrot");
    }

    private void OnTriggerExit2D(Collider2D collider)
    {
        uiShop.Hide();
    }
}
```

Table 1. Displaying shop

```
private void CreateItemButton(Sprite itemSprite, string itemName, int itemCost, int positionIndex, string displayedName)
{
    Transform shopItemTransform = Instantiate(shopItemTemplate, container);
    RectTransform shopItemRectTransform = shopItemTransform.GetComponent<RectTransform>();
    float shopItemHeight = 60f;
    shopItemRectTransform.anchoredPosition = new Vector2(0, 150 + (-shopItemHeight * positionIndex));
    shopItemTransform.Find("nameText").GetComponent<TextMeshProUGUI>().SetText(displayedName);
    shopItemTransform.Find("priceText").GetComponent<TextMeshProUGUI>().SetText(itemCost.ToString());
    shopItemTransform.Find("itemIcon").GetComponent<Image>().sprite = itemSprite;

    Item newItem = ScriptableObject.CreateInstance<Item>();

    foreach (ItemSlot itemSlot in GameManager.instance.allItemsContainer.slots)
    {
        if (itemSlot.item.Name == itemName)
        {
            newItem = itemSlot.item;
        }
    }

    btn = shopItemTransform.GetComponent<Button>();
    btn.onClick.AddListener(delegate { TaskWithParameters(itemCost, newItem); });
}
```

Table 1. Creating one item in a store

Feature #5. *Survival mechanism*

As a survival element, we implemented 4 parameters responsible for determining the state of our character. These parameters are health, hunger, temperature and time[1]. Throughout the game, the level of our hunger drops, forcing us to ensure its sources. During the night, our temperature drops rapidly, forcing us to find a source of heat, which in our case is the fire described later in the report. Both temperature and hunger are linked to health, which begins to decline when one of these values reaches the lower threshold.

Figure 1. Health, hunger, temperature display

The code snippet below describes how the hunger slump mechanism works over time.

```
//licznik wskaźnika głodu i temperatury
hungerUpdaterCounter += 1;
temperatureUpdateCounter += 1;
//tutaj dostosowac jak szybko maleje wskaznik najedzenia
if (hungerUpdaterCounter == 500)
{
    HungerController.currentHunger -= 1;
    hungerUpdaterCounter = 0;
}
//gdy wskaźnik najedzenia lub temperatury jest niższy niż 10, zaczyna ubywać zdrowia:
if(HungerController.currentHunger < 10 || TemperatureController.currentTemperature < 10)
{
    healthUpdaterCounter += 1;
    //tutaj dostosowac jak szybko maleje wskaznik zdrowia
    if (healthUpdaterCounter == 100)
    {
        HealthController.currentHealth -= 1;
        healthUpdaterCounter = 0;
    }
}
```

Table 1. Hunger decrementation

Feature #6. Night and Campfire

As another element of survival, we added night as a time in which we need to be especially careful. During the night, our body temperature begins to drop, and when it reaches a critical point, it begins to affect our health. The only way to avoid it is to stay near a burning fire. It is our only source of light at night and only in its vicinity we are able to maintain the correct temperature. To activate the bonfire, you need to spend 5 pieces of wood.

Figure 1. Active campfire during night

The code below describes the behavior of temperature and lighting depending on the current time, which is converted into hours in the game. In the range that describes the day, the temperature and lighting remain constant. The night is divided into two stages, the stage of darkening and brightening. In both of these ranges, our character's temperature drops.

```
//Światło dzienne od 4 do 20
if (time > 25200f && time < 72000f)
{
    light.intensity = 1f;
    TemperatureController.currentTemperature = 100;
}

//Gaśnie w godzinach 20 - 4
if ((time > 72000f && time < 86400f) || ((time > 0f && time < 21600f)))
{
    if (light.intensity > 0.1f)
        light.intensity -= LightTransition;
    if (temperatureUpdateCounter > 50)
        TemperatureController.currentTemperature -= 1;
        temperatureUpdateCounter = 0;
}

//Rozjaśnia się w godzinach 4 - 7
if (time > 21600f && time < 25200f)
{
    if (light.intensity < 1f)
        light.intensity += LightTransition;
    if (temperatureUpdateCounter > 50)
    {
        TemperatureController.currentTemperature -= 1;
        temperatureUpdateCounter = 0;
    }
}
```

Table2. Light and temperature change

Feature #7. *Cutting down trees*

A very important element of our game is cutting down trees. It's a necessary mechanism for the player to survive because the wood is later used to light up the campfire. You can find lots of trees on the map [1]. To cut it you need to own an axe which you can find in the starter chest and also hit it 3 times [2]. After that the logs fall out of the tree and you can collect them just by approaching them. Every cut tree increases the number of points by 40.

Figure 1. Trees on the map

Figure 2. Logs fallen out of the tree after cutting it

The code below is responsible for picking up logs from the ground by the player and adding them to the inventory. After the player is in a certain small distance from the logs they start moving towards him. The toolbar refreshes itself and we can see logs appearing in it.

```
Unity Message | Odwołania: 0
private void Update()
{
    float distance = Vector3.Distance(transform.position, player.position);

    // if the player is not in the distance to pick up logs no function is executed
    if (distance > pickUpDistance)
    {
        return;
    }

    transform.position = Vector3.MoveTowards(transform.position, player.position, speed * Time.deltaTime);

    if (distance < 0.1f)
    {
        if (GameManager.instance.inventoryContainer != null)
        {
            GameManager.instance.inventoryContainer.Add(item, count);

            toolbar.SetActive(!toolbar.activeInHierarchy);
            toolbar.SetActive(true);
        }
        else
        {
            Debug.LogWarning("no inventory container attached to game manager");
        }

        Destroy(gameObject);

    }
}
```

Table 1. Picking up logs from the ground

This code is responsible for the mechanism of cutting the tree. After the player hits the tree 3 times, 40 points are added to the current amount and then the position of the logs is being calculated. They are then being instantiated on the map and the tree is deleted from the map.

```
Odwołania: 4
public override void Hit()
{

    FindObjectOfType<SoundManager>().Play("Cut");
    hitCount++;
    if (hitCount >= 3){
        MoneyController.money += 40;
        // spawning wood
        while (dropCount > 0)
        {
            dropCount -= 1;

            // calculating where logs will drop
            Vector3 position = transform.position;
            position.x -= spread * UnityEngine.Random.value - spread / 2;
            position.y -= spread * UnityEngine.Random.value - spread / 2;
            GameObject log = Instantiate(pickUpDrop);
            log.transform.position = position;
        }

        Destroy(gameObject);
    }
}
```

Table 1. Mechanism of cutting down trees

Assets and resources used

1. Overworld tileset:
   https://opengameart.org/content/zelda-like-tilesets-and-sprites
   https://opengameart.org/content/lpc-tile-atlas
2. Tile map asset #2. *Url*
3. Character with animations *https://opengameart.org/content/zelda-like-tilesets-and-sprites*
4. Game objects graphics:
   https://opengameart.org/content/bag-icon
   https://opengameart.org/content/plants-and-flowers-pixel-art
   https://opengameart.org/content/camp-fire-animation-for-rpgs-finished
5. Sounds:
   https://freesound.org/people/Coral_Island_Studios/sounds/432591/
   https://freesound.org/people/mugwood/sounds/231757/
   https://soundbible.com/1460-Water-Splash.html
   https://soundbible.com/2204-Poker-Chips.html#google_vignette
   https://soundbible.com/290-Axe-Swing.html
   https://www.freesoundeffects.com/free-track/feuer-426782/
6. *Light 2D: Universal Render Pipeline by Unity*
7. *Background image:*
   https://pixabay.com/illustrations/island-tropical-sunset-ocean-sky-681459/
8. *Fonts: Berlin Sans, Niagara (Windows fonts)*