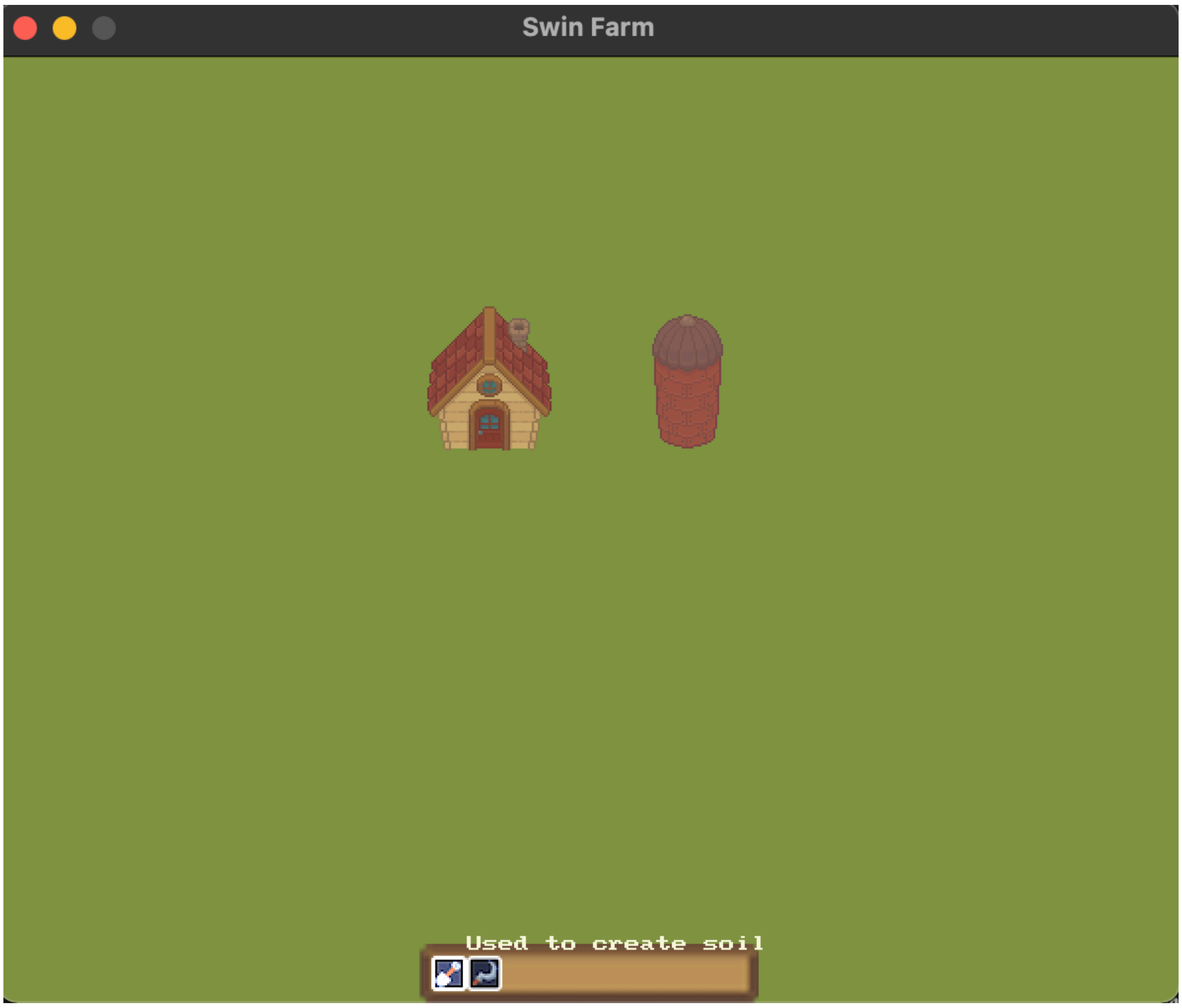
Design Overview for SwinFarm

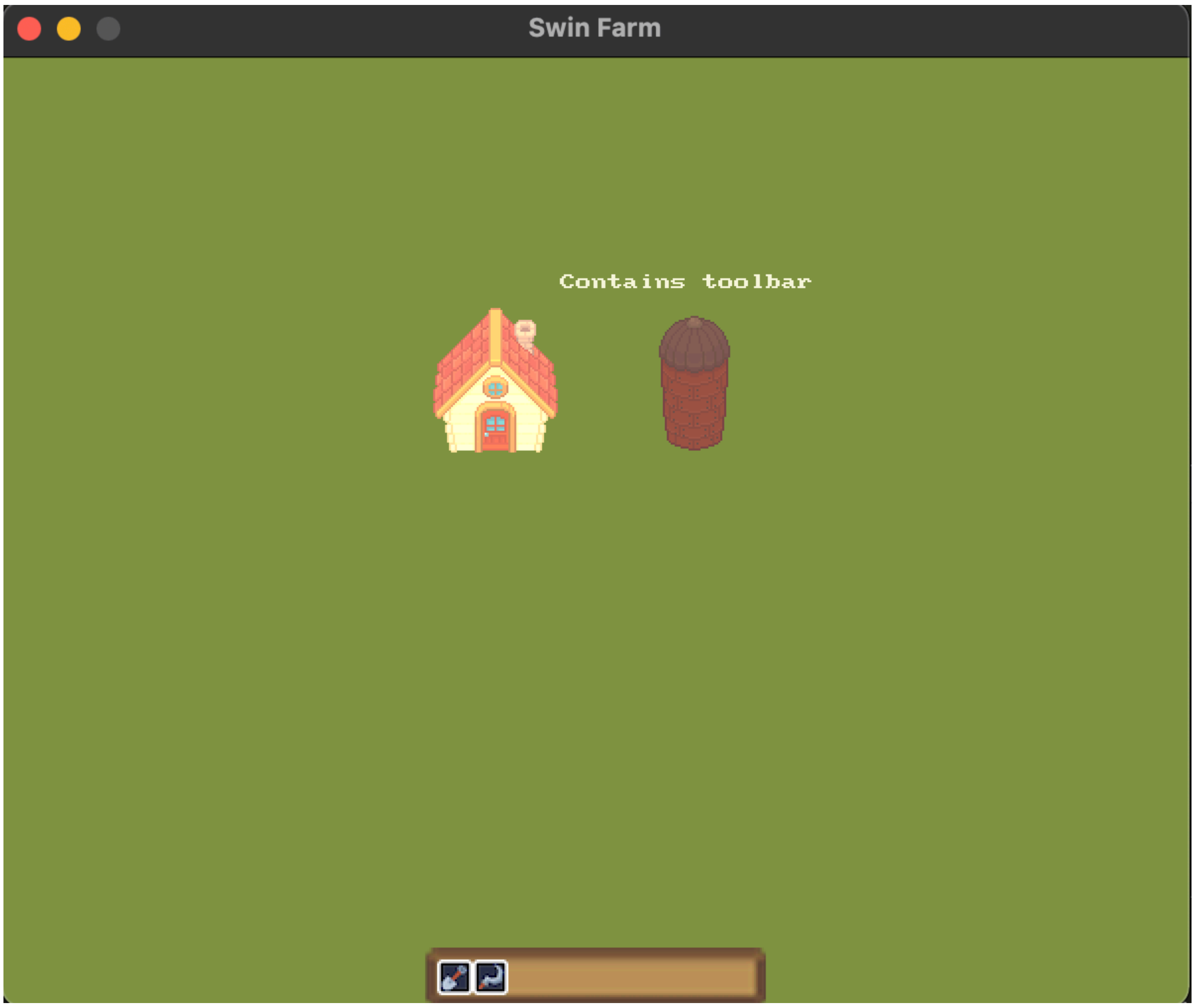
Name: Gia Bao Bui
Student ID: 103533680

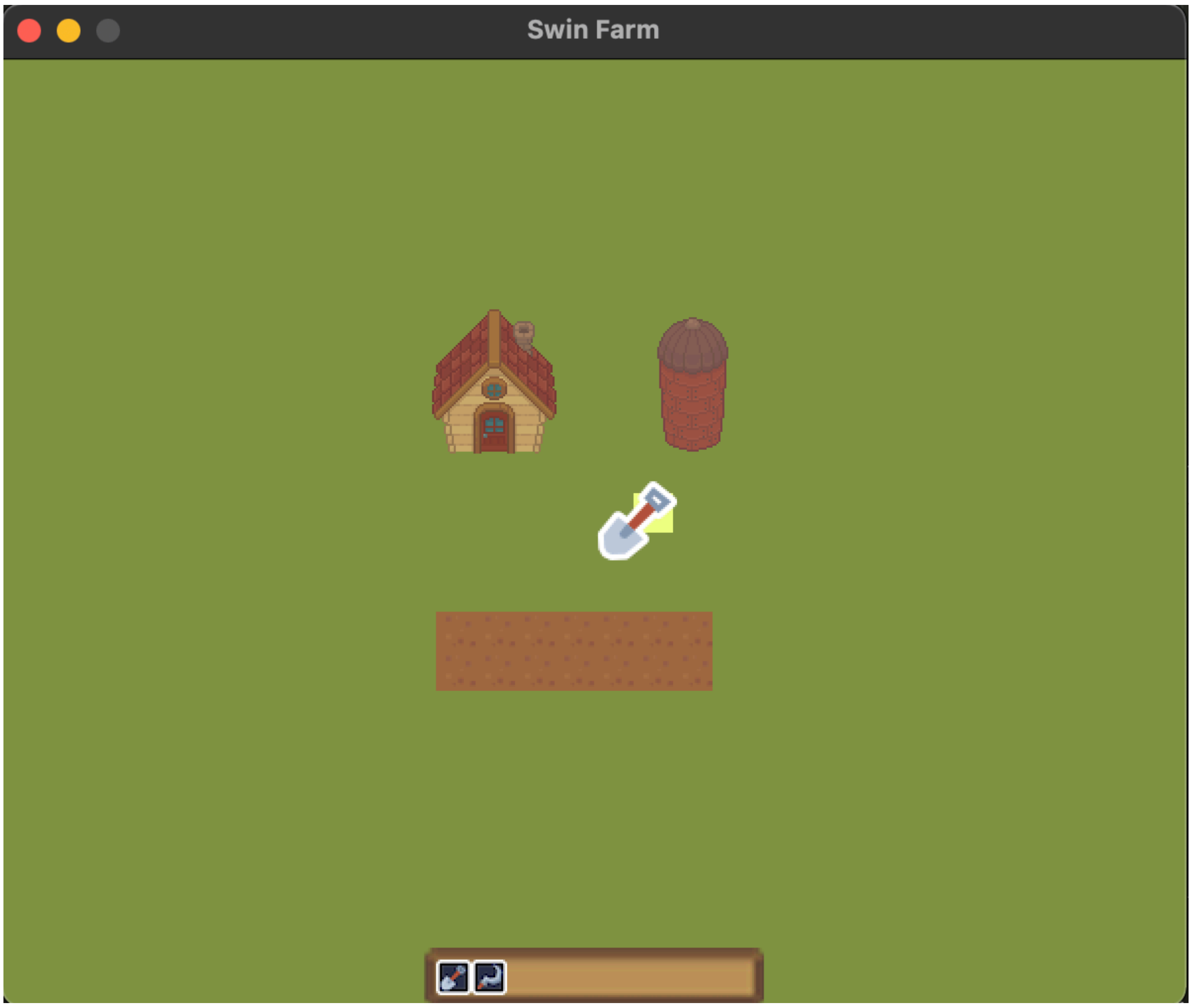
Summary of Program

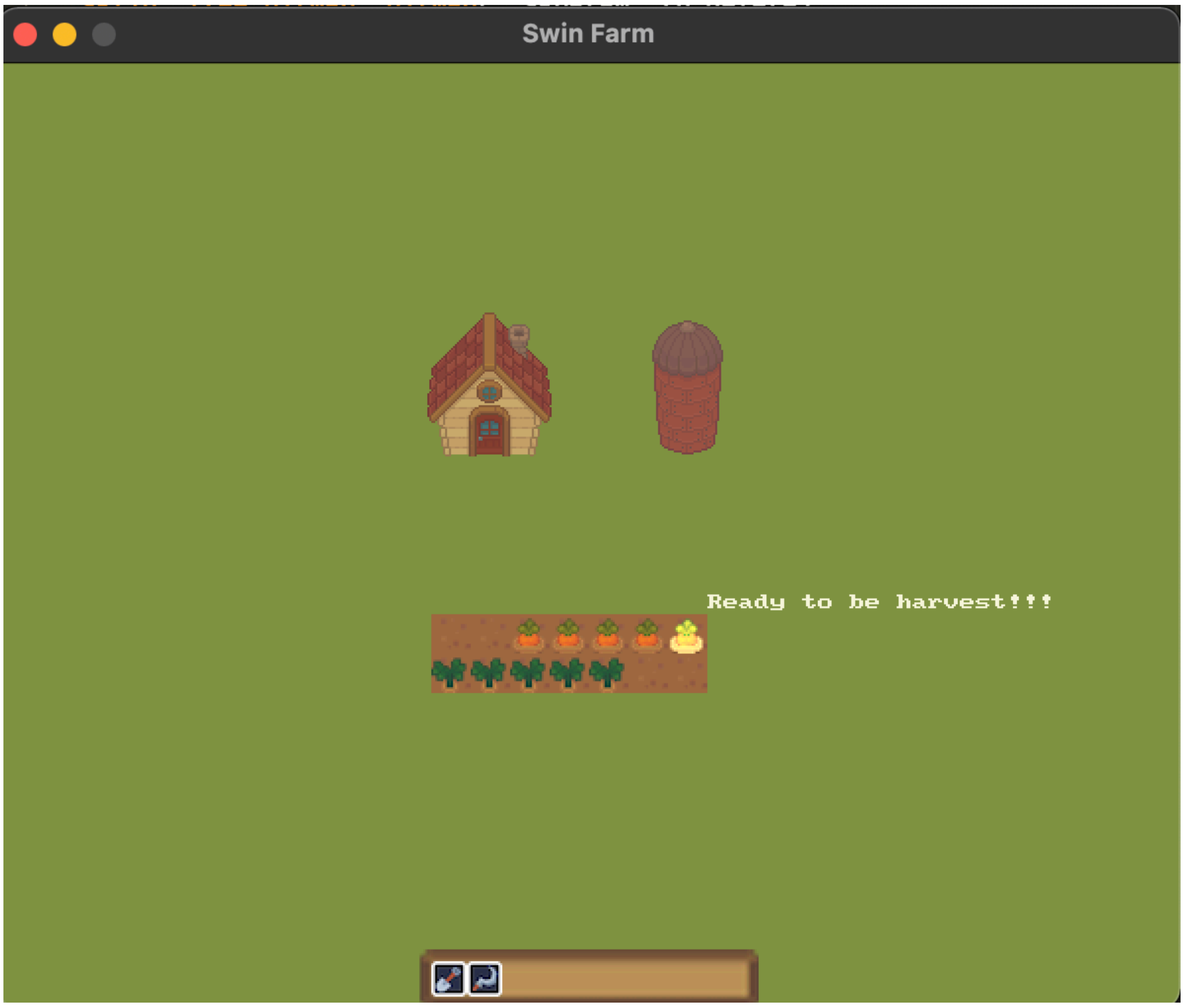
My project name is SwinFarm – a UI farming game in which players can own their small farm and make their own living. The initial requirements of the game are users can grow, harvest, and store their produce. The game will continue to be upgraded for more features if possible.

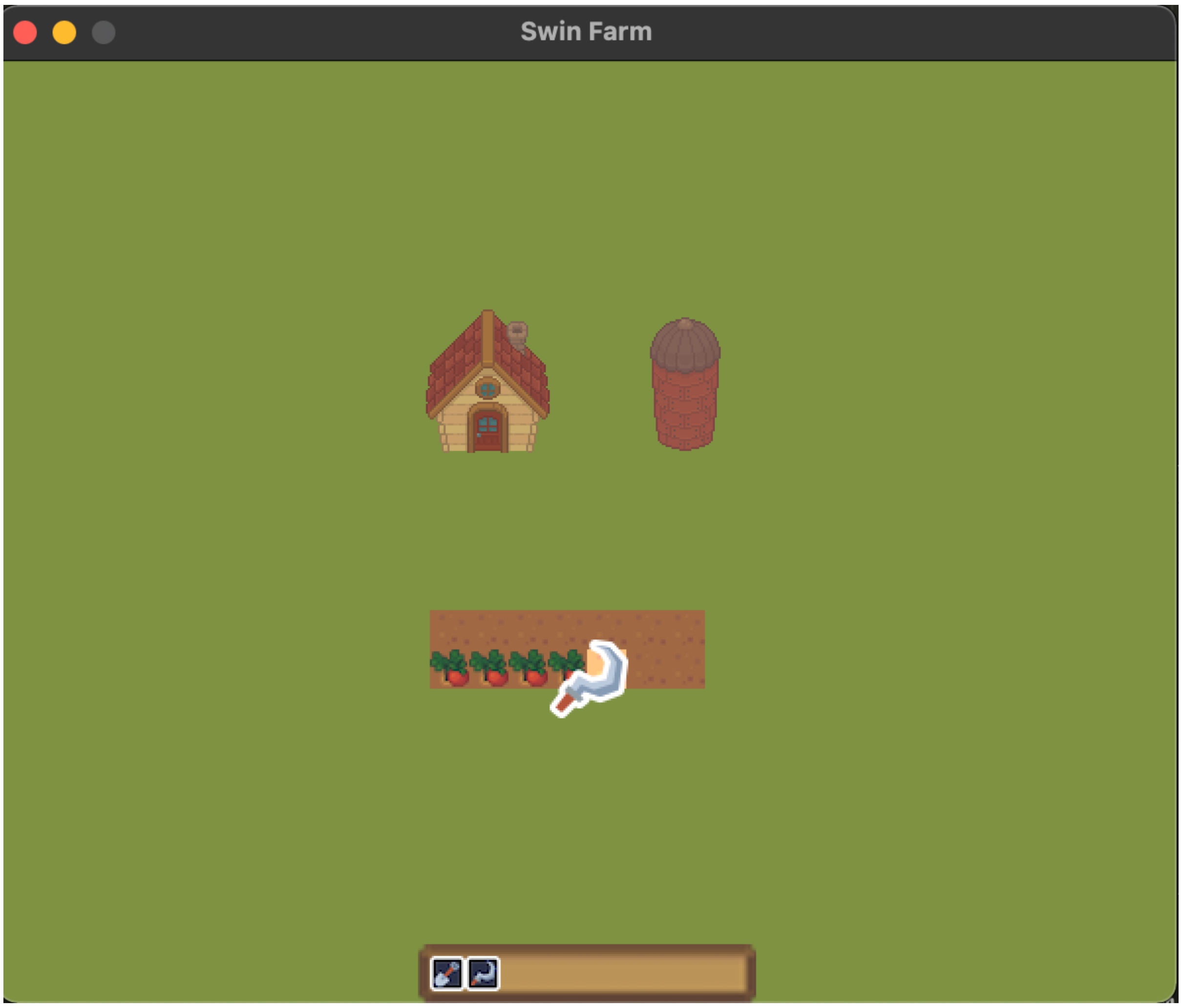
Screenshots of the program:

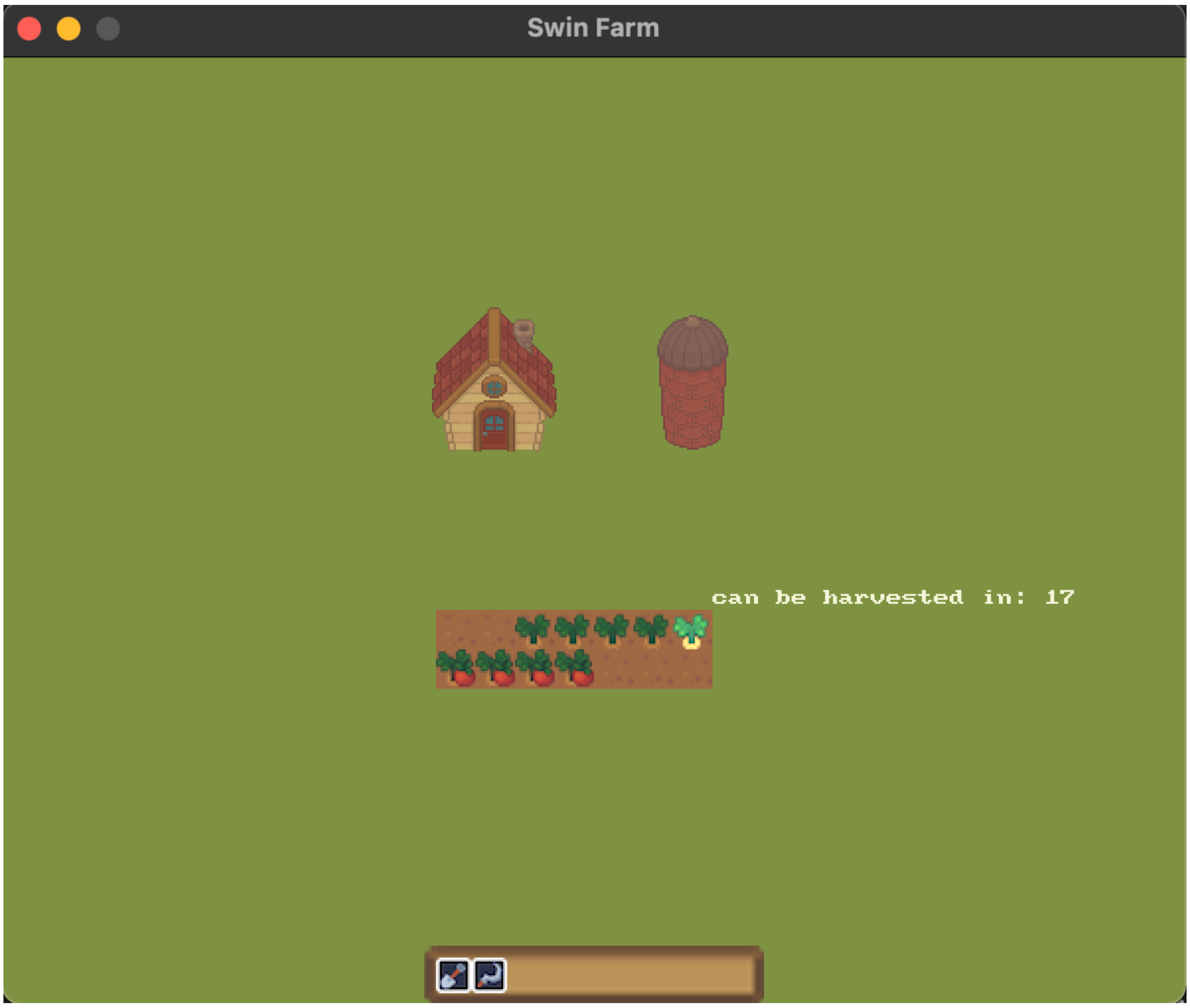


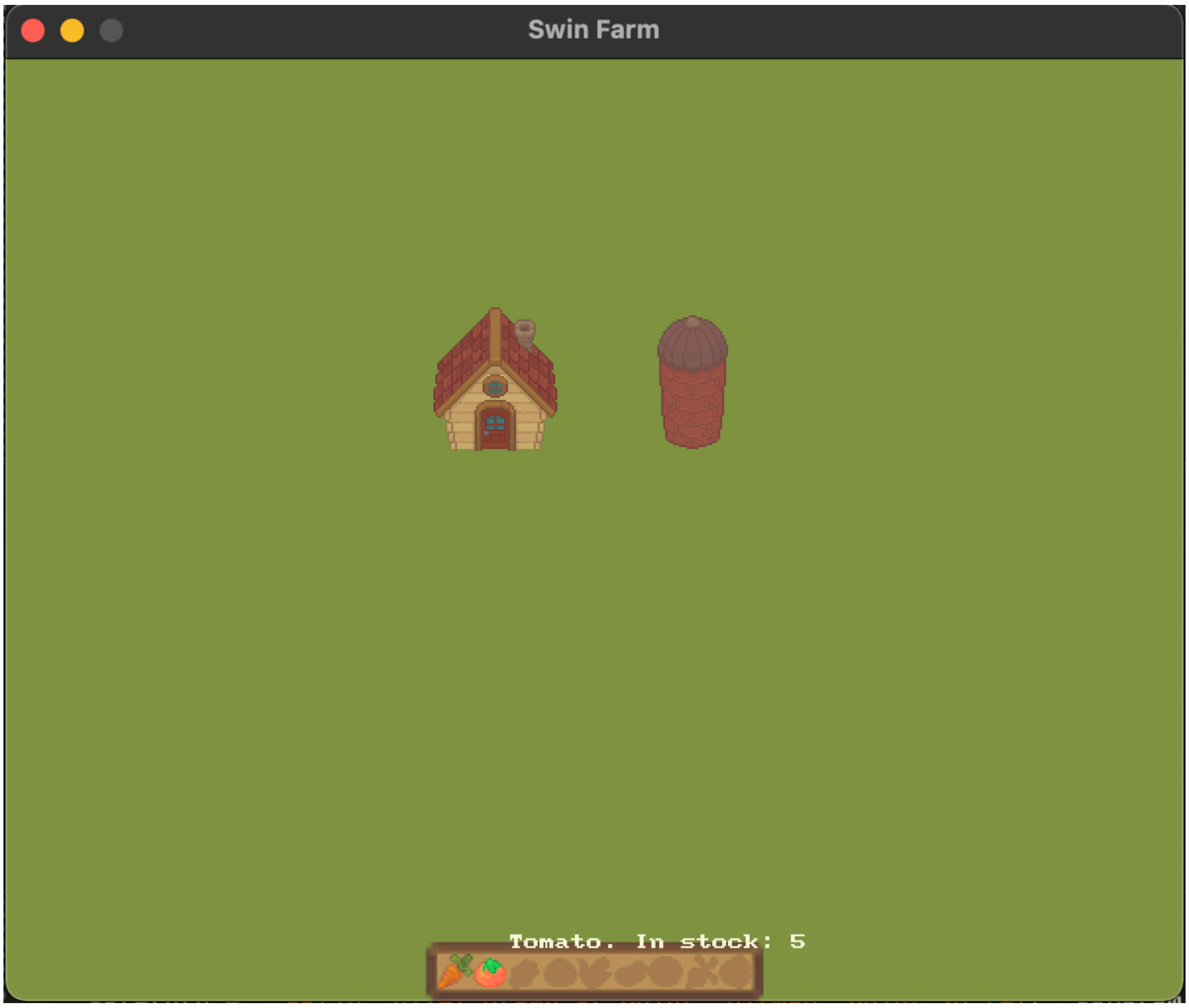

Swin Farm
Contains toolbar


Swin Farm


Swin Farm
Ready to be harvest!!!


Swin Farm


Swin Farm
can be harvested in: 17



Required Roles

Table 1: <<GameObject>> details – duplicate

Responsibility	Type Details	Notes
• Appears on the game (be drawn) • Knows whether the mouse is over or hovered for an amount of time	Has a string ID Has a string description Has an Image & its hovered version (Bitmap) Has coordination on the screen (Point2d)	All implements IDrawable, IReferable For entities that are drawn on the screen and can be located by mouse.

Table 2: <<Building>> details – duplicate

Responsibility	Type Details	Notes
• Store a type of Inventory	Has an Inventory	Inherits from GameObject

• Open the Inventory		

Table 3: <<Inventory>> details – duplicate

Responsibility	Type Details	Notes
• Contains slot to put items in • Check the "picked" status of the items.	Has a list of Slot for item's coordination definition & items' control Has a Boolean to check whether it is opened or not	Inherits from GameObject

Table 4: <<Toolbar>> details – duplicate

Responsibility	Type Details	Notes
• Contains tools	List<Tool>	Inherits from Inventory

Table 5: <<Stock>> details – duplicate

Responsibility	Type Details	Notes
• Contains produces	List<Produce>	• Inherits from Inventory • This Produce list is static to be accessed by the Expanse to update the correct Produce by the harvested Seed

Table 6: <<Item>> details – duplicate

Responsibility	Type Details	Notes
• Can be picked and unpicked	Has an image for Picked status (Bitmap) Has a Boolean to check whether it is picked or not	Inherits from GameObject

Table 7: <<Tool>> details – duplicate

Responsibility	Type Details	Notes
• Has a function		Inherits from Item

Table 8: <<Shovel>> details – duplicate

Responsibility	Type Details	Notes
• Generate a Soil (for future planting)		Inherits from Tool

Table 9: <<Sickle>> details – duplicate

Responsibility	Type Details	Notes
• Harvest the seeds		Inherits from Tool

Table 10: <<Produce>> details – duplicate

Responsibility	Type Details	Notes
• Produces the seed • Keeps track of the number of seeds in stock	An integer to keep track of the amount of stock An image of the type of Seed An image of its Seed (growing seed) & its hovered version.	Inherits from Item Produce is considered as SeedGenerator

Table 11: <<Seed>> details – duplicate

Responsibility	Type Details	Notes
• Grows by itself	A specific integer for the total time growing (uint) A starting time point (uint) A Boolean to check whether it is ready to be harvested	Inherits from GameObject

Table 12: <<Expanse>> details – duplicate

Responsibility	Type Details	Notes
• Generates all the cells (tiles: grass, soil) & restricted cells (users cannot shovel these cells) • Keeps track of the cells and soils	Images for grass/hovered grass Images for soil/hovered soil A list of Cell A list of Soil (for quick access)	Inherits from GameObject Expanse is considered as the gateway for anything related to the planting and growing produces.

Table 13: <<Cell>> details – duplicate

Responsibility	Type Details	Notes
• Determines the coordination for different functionalities across the game (optimize storage efficiency) • Has no graphic illustrations • Depending on the type of cell on the Expanse, different terrain will be visualized	Has coordinations A Boolean to check whether it is restricted	Inherits from IReferable Cell is to assist locating objects on the screen, which must be created automatically instead of being specifically designed.

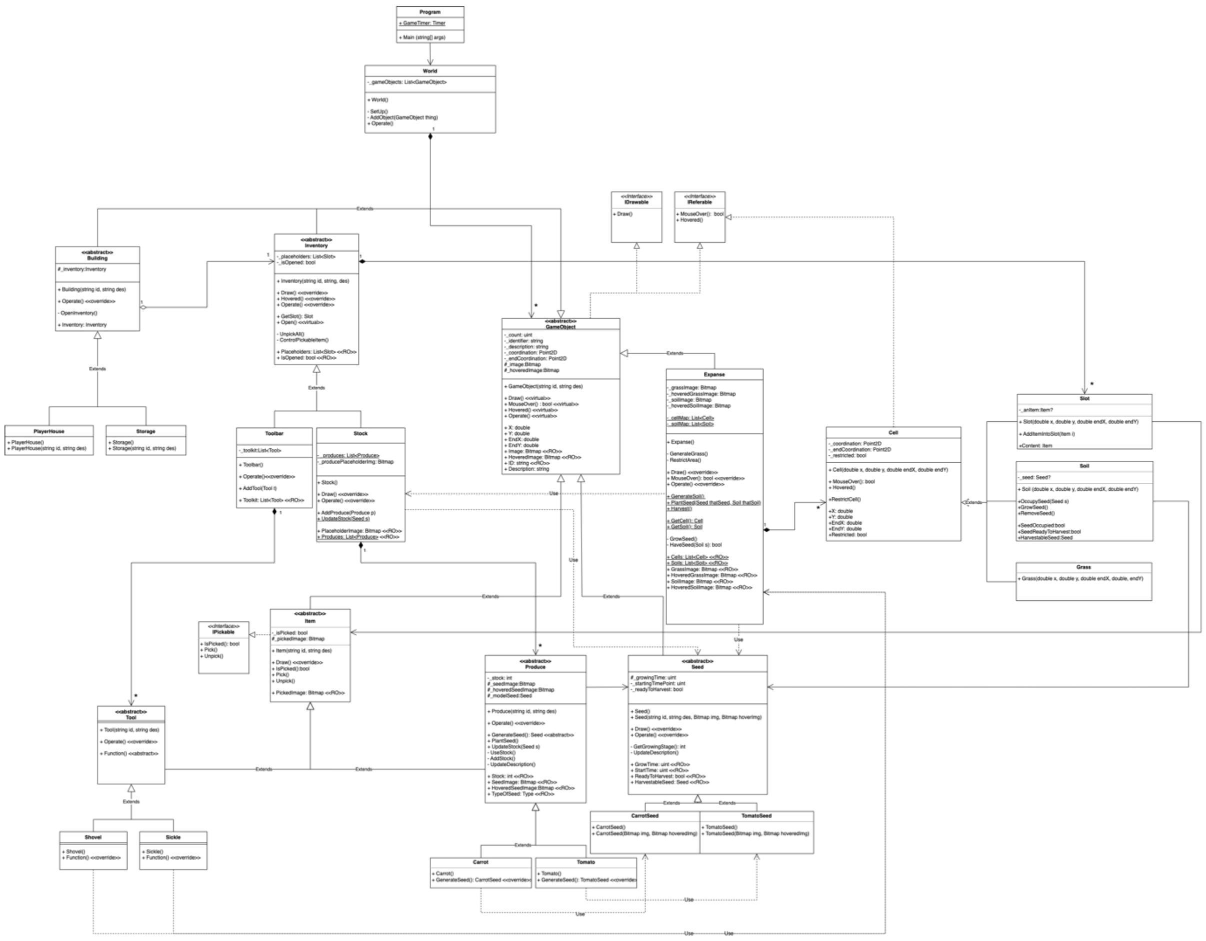
Table 13: <<Soil>> details – duplicate

Responsibility	Type Details	Notes
• Bonds with a seed • Grows the seed and returns it	A Seed (which can be null – have no seeds yet)	Inherits from Cell

Table 13: <<Slot>> details – duplicate

Responsibility	Type Details	Notes
• Bonds with an Item to locate that on the screen • Helps the inventory control "picked" state of items	An Item	Inherits from Cell

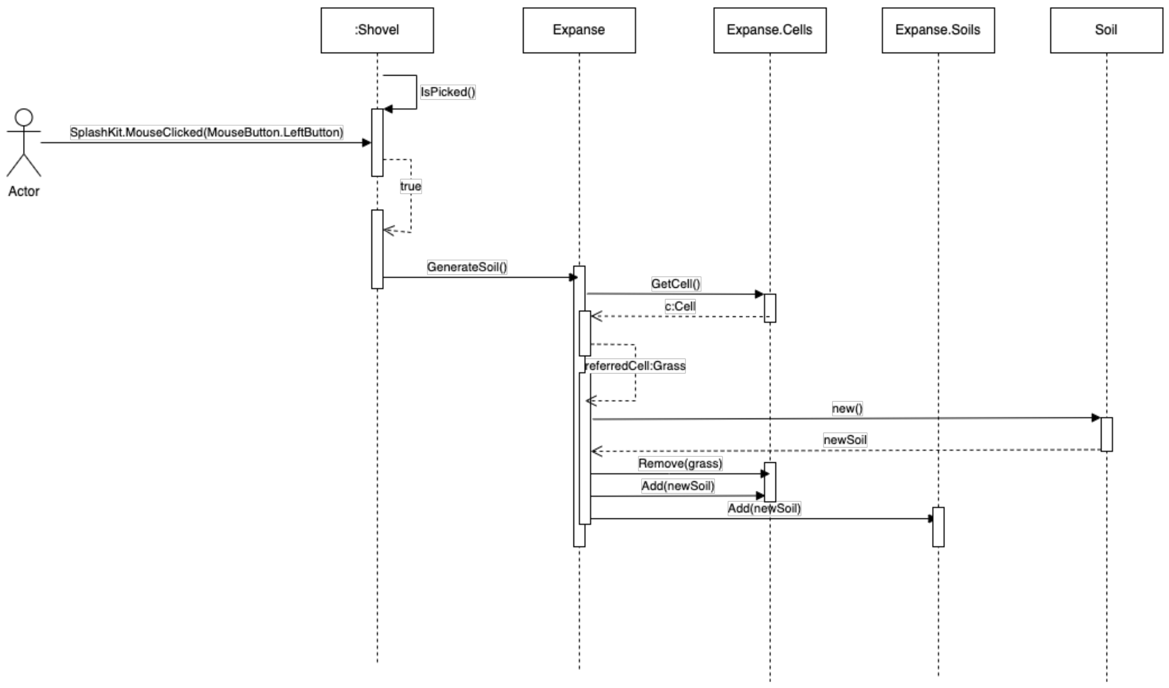
Class Diagram



Sequence Diagram

Generate a Soil to allow planting a seed in

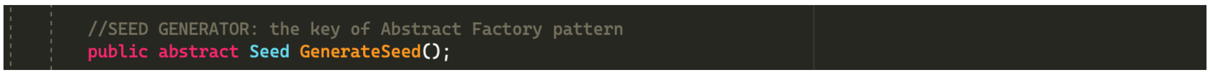
Shovel is used to meet this functionality. Shovel calls the function in Expanse (generate soil). the Expanse will loop over all its Cell and return a Cell object. If this Cell is not null and is a Grass object. Expanse removes the Grass Cell out of all the Cells it contains and adds a new Soil object in with the same location as the removed Grass.



Design patterns implementation

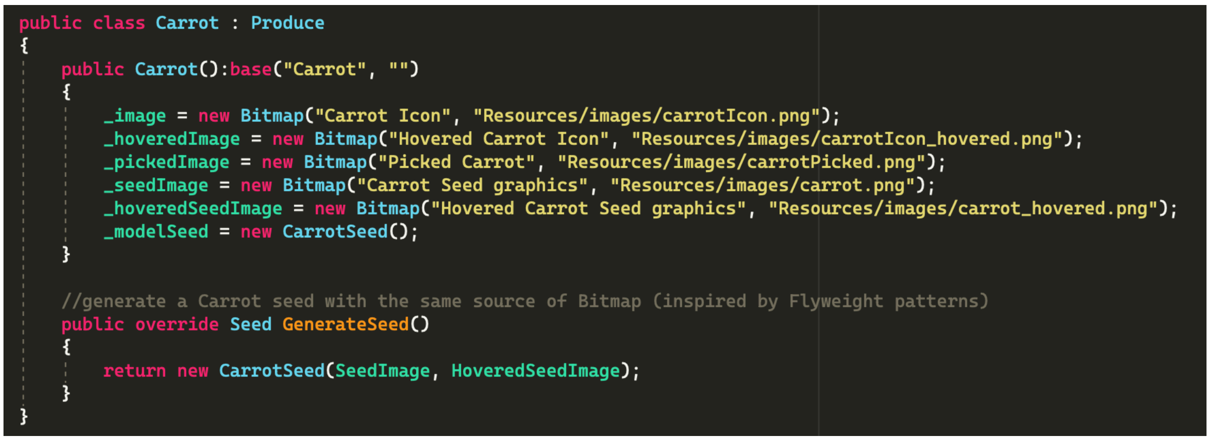
Abstract Factory

The use of Abstract Factory in my design is the collaboration between Produce and Seed. In other words, Produce can be considered a Seed Factory where its subclass can create a different product (Carrot produces CarrotSeed; Tomato produces TomatoSeed). The whole mechanism of planting and harvesting between types of Produce is the same. Therefore, the code used when the game needs to have more types of Produce can be reused.



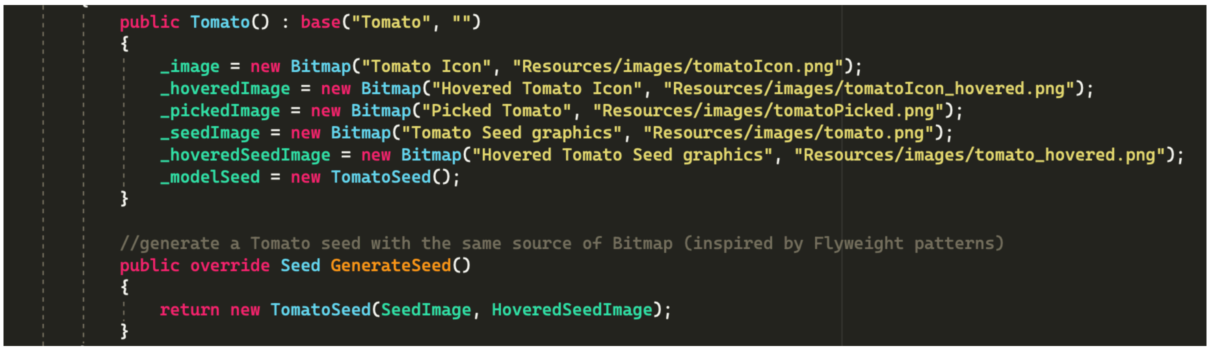
```
//SEED GENERATOR: the key of Abstract Factory pattern
public abstract Seed GenerateSeed();
```

Every Produce object has the function to produce a seed



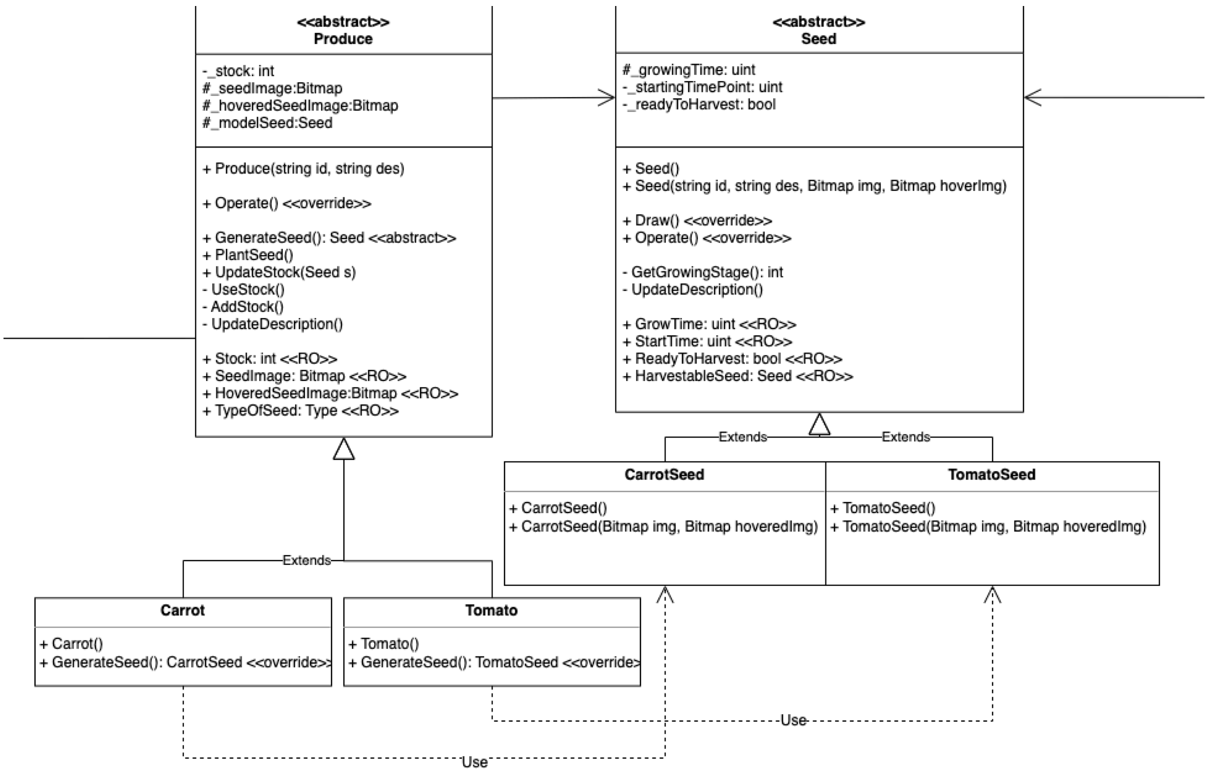
```csharp
public class Carrot : Produce
{
    public Carrot():base("Carrot", "")
    {
        _image = new Bitmap("Carrot Icon", "Resources/images/carrotIcon.png");
        _hoveredImage = new Bitmap("Hovered Carrot Icon", "Resources/images/carrotIcon_hovered.png");
        _pickedImage = new Bitmap("Picked Carrot", "Resources/images/carrotPicked.png");
        _seedImage = new Bitmap("Carrot Seed graphics", "Resources/images/carrot.png");
        _hoveredSeedImage = new Bitmap("Hovered Carrot Seed graphics", "Resources/images/carrot_hovered.png");
        _modelSeed = new CarrotSeed();
    }

    //generate a Carrot seed with the same source of Bitmap (inspired by Flyweight patterns)
    public override Seed GenerateSeed()
    {
        return new CarrotSeed(SeedImage, HoveredSeedImage);
    }
}
```

The Carrot Produce will produce a CarrotSeed



```csharp
        public Tomato() : base("Tomato", "")
        {
            _image = new Bitmap("Tomato Icon", "Resources/images/tomatoIcon.png");
            _hoveredImage = new Bitmap("Hovered Tomato Icon", "Resources/images/tomatoIcon_hovered.png");
            _pickedImage = new Bitmap("Picked Tomato", "Resources/images/tomatoPicked.png");
            _seedImage = new Bitmap("Tomato Seed graphics", "Resources/images/tomato.png");
            _hoveredSeedImage = new Bitmap("Hovered Tomato Seed graphics", "Resources/images/tomato_hovered.png");
            _modelSeed = new TomatoSeed();
        }

        //generate a Tomato seed with the same source of Bitmap (inspired by Flyweight patterns)
        public override Seed GenerateSeed()
        {
            return new TomatoSeed(SeedImage, HoveredSeedImage);
        }
```

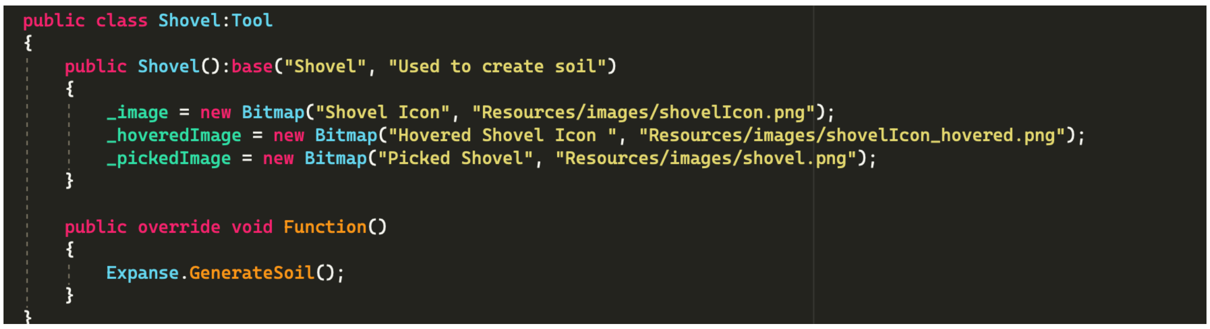
The Tomato Produce will produce a TomatoSeed



Abstract Factory implementation in SwinFarm

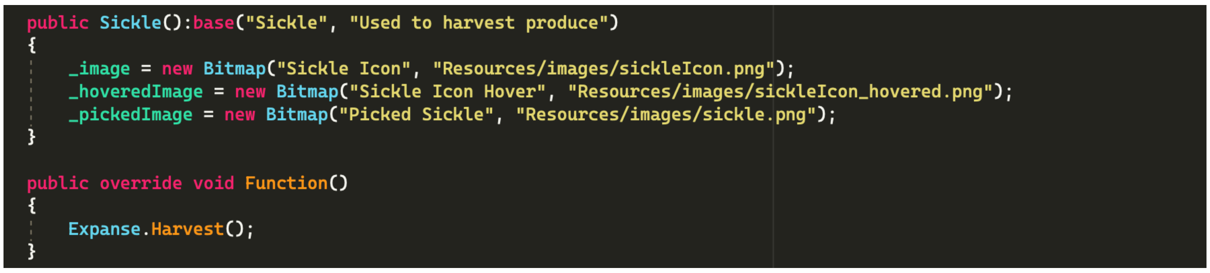
Singleton

The use of Singleton as a static class is not definitively implemented in my design. However, to enable the science of growing a tree, there are static methods in the Expanse class that can be used to offer interfaces to other classes. For example, a Shovel changes the Grass(Cell) to Soil(Cell); Produce locates Soil(Cell) and generates a Seed into that Soil via Expanse static methods.



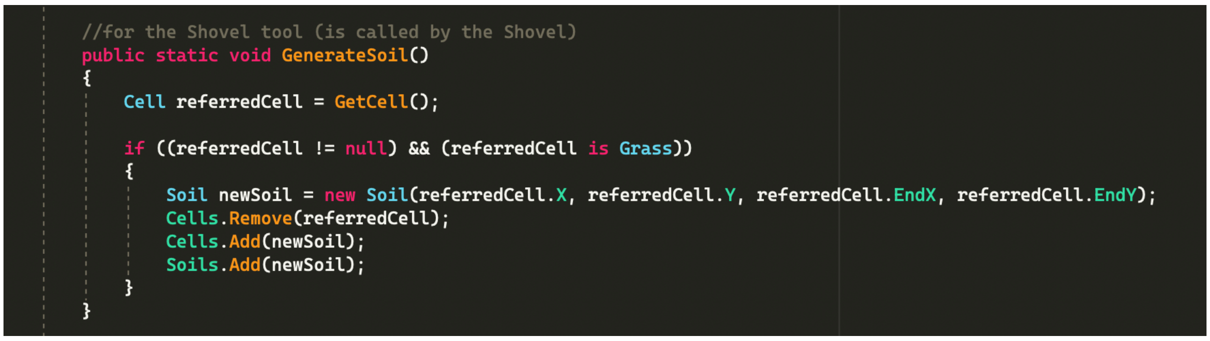
```
public class Shovel:Tool
{
    public Shovel():base("Shovel", "Used to create soil")
    {
        _image = new Bitmap("Shovel Icon", "Resources/images/shovelIcon.png");
        _hoveredImage = new Bitmap("Hovered Shovel Icon ", "Resources/images/shovelIcon_hovered.png");
        _pickedImage = new Bitmap("Picked Shovel", "Resources/images/shovel.png");
    }

    public override void Function()
    {
        Expanse.GenerateSoil();
    }
}
```

The Shovel makes use of the Expanse interface to Generate Soil



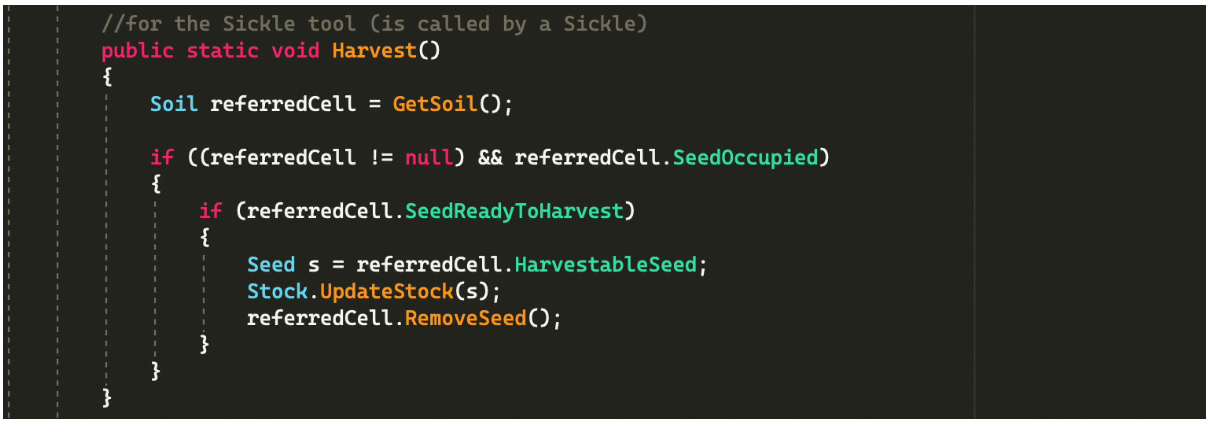
```
public Sickle():base("Sickle", "Used to harvest produce")
{
    _image = new Bitmap("Sickle Icon", "Resources/images/sickleIcon.png");
    _hoveredImage = new Bitmap("Sickle Icon Hover", "Resources/images/sickleIcon_hovered.png");
    _pickedImage = new Bitmap("Picked Sickle", "Resources/images/sickle.png");
}

public override void Function()
{
    Expanse.Harvest();
}
```

The Sickle makes use of the Expanse interface to Harvest produce



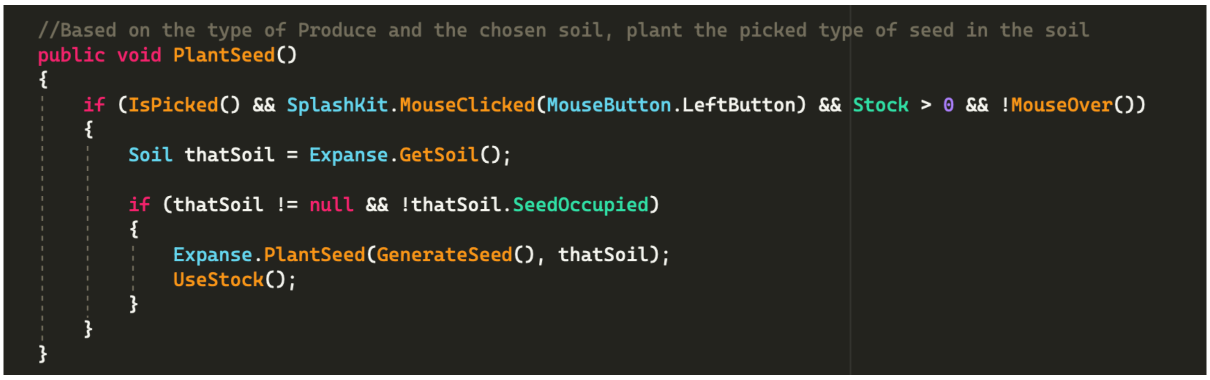
```
    //for the Shovel tool (is called by the Shovel)
    public static void GenerateSoil()
    {
        Cell referredCell = GetCell();

        if ((referredCell != null) && (referredCell is Grass))
        {
            Soil newSoil = new Soil(referredCell.X, referredCell.Y, referredCell.EndX, referredCell.EndY);
            Cells.Remove(referredCell);
            Cells.Add(newSoil);
            Soils.Add(newSoil);
        }
    }
```

Expanse provides an interface for Shovel to achieve its function



```
//for the Sickle tool (is called by a Sickle)
public static void Harvest()
{
    Soil referredCell = GetSoil();

    if ((referredCell != null) && referredCell.SeedOccupied)
    {
        if (referredCell.SeedReadyToHarvest)
        {
            Seed s = referredCell.HarvestableSeed;
            Stock.UpdateStock(s);
            referredCell.RemoveSeed();
        }
    }
}
```

Expanse provides an interface for Sickle to achieve its function

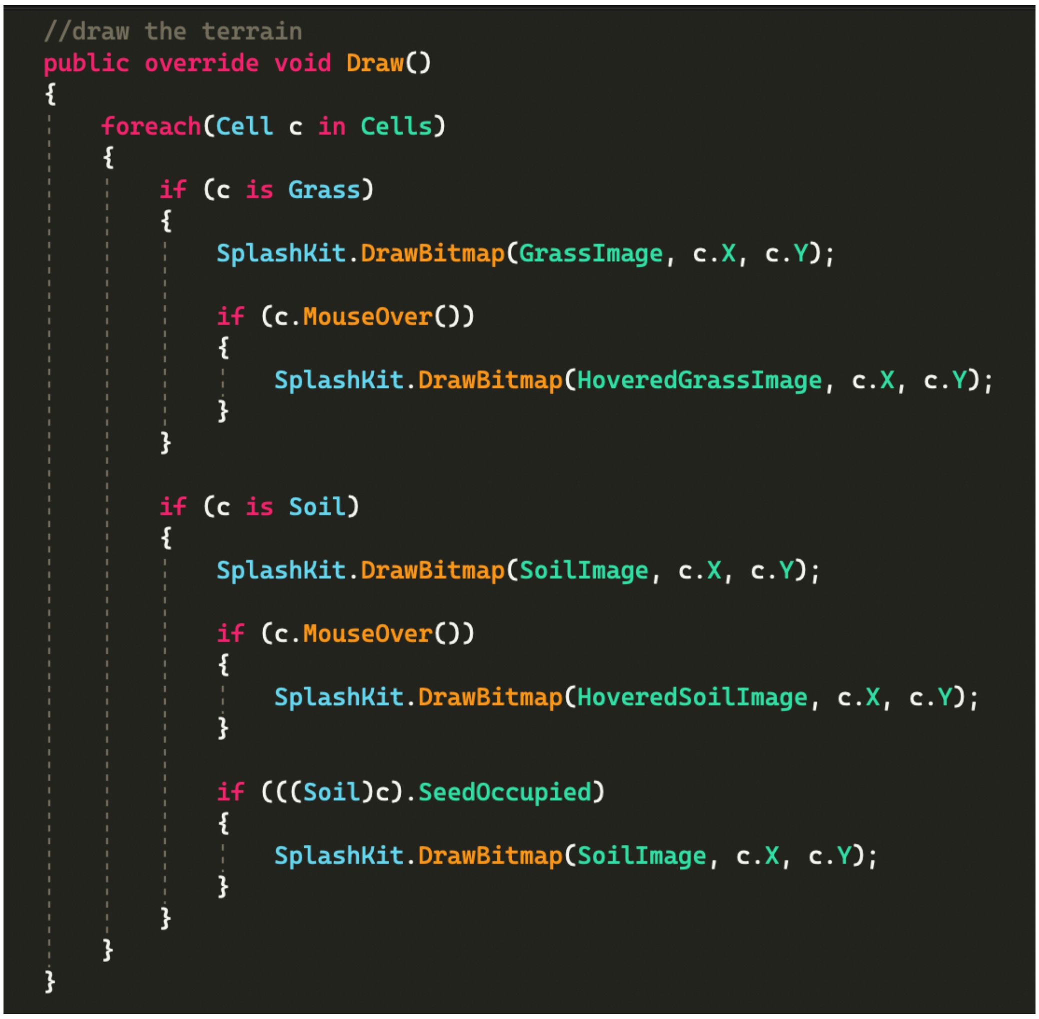


```
//Based on the type of Produce and the chosen soil, plant the picked type of seed in the soil
public void PlantSeed()
{
    if (IsPicked() && SplashKit.MouseClicked(MouseButton.LeftButton) && Stock > 0 && !MouseOver())
    {
        Soil thatSoil = Expanse.GetSoil();

        if (thatSoil != null && !thatSoil.SeedOccupied)
        {
            Expanse.PlantSeed(GenerateSeed(), thatSoil);
            UseStock();
        }
    }
}
```

Produce objects will call the PlantSeed interface of Expanse to plant their seed into a soil

Flyweight

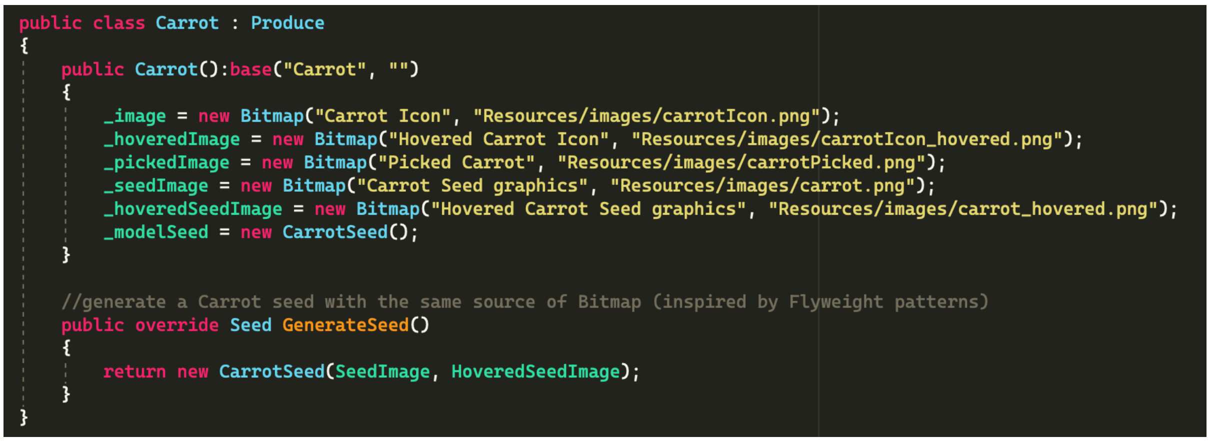
In hindsight, the design of my program is significantly inspired by flyweight patterns. There are multiple incidents when a Bitmap's usage is considered because the Flyweight pattern is all about making the design lightweight. For example, instead of storing a Bitmap of grass & hovered grass in the Grass class, the Expanse class has been dedicated to storing one instance of these Bitmaps and drawing it depending on the type of Cell (either Grass or Soil). Otherwise, when creating the whole grid of terrain of grass, multiple instances of Bitmap grass would be created, which renders the program cumbersome.



```
//draw the terrain
public override void Draw()
{
    foreach(Cell c in Cells)
    {
        if (c is Grass)
        {
            SplashKit.DrawBitmap(GrassImage, c.X, c.Y);

            if (c.MouseOver())
            {
                SplashKit.DrawBitmap(HoveredGrassImage, c.X, c.Y);
            }
        }

        if (c is Soil)
        {
            SplashKit.DrawBitmap(SoilImage, c.X, c.Y);

            if (c.MouseOver())
            {
                SplashKit.DrawBitmap(HoveredSoilImage, c.X, c.Y);
            }

            if (((Soil)c).SeedOccupied)
            {
                SplashKit.DrawBitmap(SoilImage, c.X, c.Y);
            }
        }
    }
}
```

Expanse uses its own instance of Bitmap to draw grass and soil.

In addition, the Produce class, and subclasses (Carrot, Tomato) all store their own Bitmap of the seed it produces. Therefore, whenever the seed is generated, it will refer to the same Bitmap of the type of its Produce.



```
public class Carrot : Produce
{
    public Carrot():base("Carrot", "")
    {
        _image = new Bitmap("Carrot Icon", "Resources/images/carrotIcon.png");
        _hoveredImage = new Bitmap("Hovered Carrot Icon", "Resources/images/carrotIcon_hovered.png");
        _pickedImage = new Bitmap("Picked Carrot", "Resources/images/carrotPicked.png");
        _seedImage = new Bitmap("Carrot Seed graphics", "Resources/images/carrot.png");
        _hoveredSeedImage = new Bitmap("Hovered Carrot Seed graphics", "Resources/images/carrot_hovered.png");
        _modelSeed = new CarrotSeed();
    }

    //generate a Carrot seed with the same source of Bitmap (inspired by Flyweight patterns)
    public override Seed GenerateSeed()
    {
        return new CarrotSeed(SeedImage, HoveredSeedImage);
    }
}
```

The new CarrotSeed objects that will be generated will refer to only one instance of Bitmap located in its Produce (its Factory).