Stay # - Snake

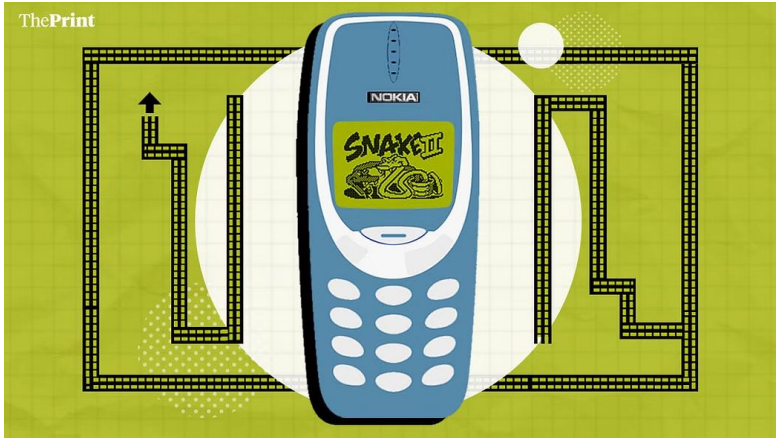
4-H GROWS HERE

Code Camp 2023

Jackson’s Mill

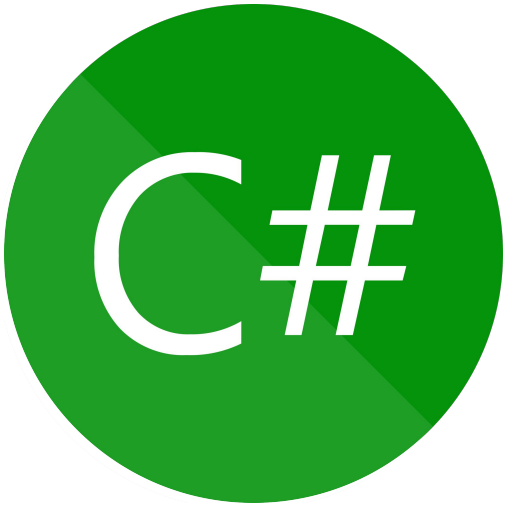
4-H GROWS HERE

Overview



- Snake can trace its origins to the 1976 arcade game Blockade. It can be argued however that the concept did not become a cultural touchstone until the release of Snake in 1997 on the Nokia 6110.
- Since then a great deal has changed and the game has been ported to multiple platforms.
- Today you will help us build a version that leverages C#.

What we are making?



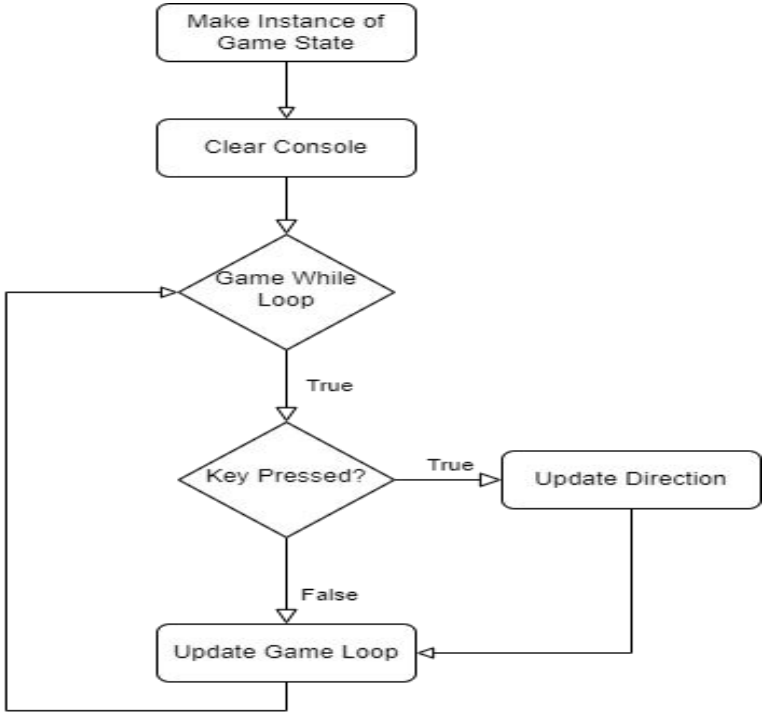
Setup

git clone -b CodeCamp2023 https://github.com/markghareeb/code-camp.git

Sprint Goal

- Develop and Test these methods
  - Public void UpdateDirection()
  - Public CalculateNewHead()
  - Public bool GameOverCollision()
  - Public bool WillTheSnakeEatAnApple()

The Magic of Modeling



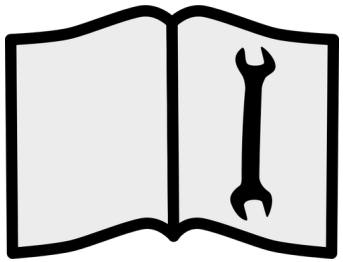
- Modeling can be an important part of the development process.
- Once a project goes beyond the scope of a simple program having a plan or some sort of structure can guide the process and minimize errors.
- This can be thought of in the same way as an english paper. You write an outline before to organize the large task.

Method Walkthrough and Syntax

```
public void DoGameLoop(bool moveOverride)
{
    if (gameTimer.ElapsedMilliseconds >= Speed || moveOverride)
    {
        if (GameOverCollision())
        {
            Console.Beep();
            Console.WriteLine("YOU LOSE");
            Environment.Exit(0);
        }

        var appleEaten = WillTheSnakeEatAnApple();
        if (appleEaten)
        {
            ApplePosition = GetNewApplePosition();
            Score += Speed;
            Console.Beep();
        }
        Snake.Move(appleEaten);

        Console.SetCursorPosition(0, 0);
        DrawBoard();
        Console.WriteLine(Score);
        gameTimer.Restart();
    }
}
```



UpdateDirection()

Update Direction - Outline

What the method needs to do: `public void UpdateDirection(ConsoleKeyInfo keyPressed)`

- Update the value of the variable that controls the direction the snake moves with the value passed in from the keyboard. Be mindful of erroneous input.

A breakdown of what needs to be done:

- Guard for erroneous input
- Determine the new direction
- Update the control variable

Update Direction - Answer

```
1 reference
public void UpdateDirection(ConsoleKeyInfo keyPressed)
{
    if (keyPressed.Key == ConsoleKey.UpArrow && previousDirection == Direction.Down ||
        keyPressed.Key == ConsoleKey.RightArrow && previousDirection == Direction.Left ||
        keyPressed.Key == ConsoleKey.DownArrow && previousDirection == Direction.Up ||
        keyPressed.Key == ConsoleKey.LeftArrow && previousDirection == Direction.Right)
    {
        return;
    }

    if (keyPressed.Key == ConsoleKey.UpArrow)
    {
        Direction = Direction.Up;
    }
    else if (keyPressed.Key == ConsoleKey.RightArrow)
    {
        Direction = Direction.Right;
    }
    else if (keyPressed.Key == ConsoleKey.DownArrow)
    {
        Direction = Direction.Down;
    }
    else if(keyPressed.Key == ConsoleKey.LeftArrow)
    {
        Direction = Direction.Left;
    }
}
```

CalculateNewHead()

CalculateNewHead - Outline

What the method needs to do: `public (int x, int y) CalculateNewHead()`

- Based on the direction the snake needs to move, update the cartesian coordinate that represents its position on the game board.

A breakdown of what needs to be done:

- Get the position of the snake
- Get the direction the snake is to move in
- Calculate the new position
- Return the newly calculated position

CalculateNewHead - Answer

```
public (int x, int y) CalculateNewHead()
{
    (int x, int y) newHeadPosition;
    if (Direction == Direction.Up)
    {
        newHeadPosition = (Head.Value.x - 1, Head.Value.y);
    }
    else if (Direction == Direction.Right)
    {
        newHeadPosition = (Head.Value.x, Head.Value.y + 1);
    }
    else if (Direction == Direction.Down)
    {
        newHeadPosition = (Head.Value.x + 1, Head.Value.y);
    }
    else
    {
        newHeadPosition = (Head.Value.x, Head.Value.y - 1);
    }
    return newHeadPosition;
}
```

GameOverCollision()

GameOverCollision- Outline

What the method needs to do: `public bool GameOverCollision()`

- This method needs to determine if a collision will occur, based on the next position.

A breakdown of what needs to be done:

- Get the next head position
- Check if the position is the same as a wall
- Check if the position is the same as the snake body
- Return the status

GameOverCollision- Answer

```
1 reference
public bool GameOverCollision()
{
    var nextHeadPosition = Snake.CalculateNewHead();
    return WallHit(nextHeadPosition) || BodyHit(nextHeadPosition);
}
```

WillTheSnakeEatAnApple()

WillTheSnakeEatAnApple - Outline

What the method needs to do: `public bool WillTheSnakeEatAnApple()`

- This method needs to determine if a collision, with the apple, will occur based on the next position.

A breakdown of what needs to be done:

- Get next position of the snake head
- Get the position of the apple
- Check if they are the same
- Return the answer

WillTheSnakeEatAnApple - Answer

```
1 reference
public bool WillTheSnakeEatAnApple()
{
    var nextHeadPosition = Snake.CalculateNewHead();
    return nextHeadPosition == ApplePosition;
}
```

GetNewApplePosition()

GetNewApplePosition - Outline

What the method needs to do: `public (int x, int y) GetNewApplePosition()`

- When a game starts or when the snake eats an apple, the apple needs to move. This method needs to return a valid apple position based on the current position of the snake and the dimensions of the game board.

A breakdown of what needs to be done:

- Get bounds of game board
- Get snake position
- Generate random apple position
- Make sure the new apple won't be in the snake or the wall
- Return apple position

GetNewApplePosition - Answer

```
2 references
public (int x, int y) GetNewApplePosition()
{
    (int x, int y) applePosition;
    do
    {
        applePosition = (Random.Shared.Next(1, bottomWallPosition - 1), Random.Shared.Next(1, rightWallPosition - 1));
    } while (Snake.IsHere(applePosition));
    return applePosition;
}
```

Congratulations!

…Time for the bonus!