

ГБОУ "Президентский ФМЛ № 239"
«Создание игры – лабиринт “ШОНДИ”»

Годовой проект по информатике

Ольховская Камилла, 10–3 класс

Постановка задачи

Создать игру – лабиринт на платформе Unity на языке C#.

Необходимо создать игру, в которой персонаж благодаря нажатию на кнопки клавиатуры будет передвигаться по лабиринту, собирая очки за каждую пройденную единицу пути.

Реализация набора очков будет происходить через столкновение с объектами – конфетами и начисления за это очков.

Уточнение исходных и выходных данных

Исходными данными являются:

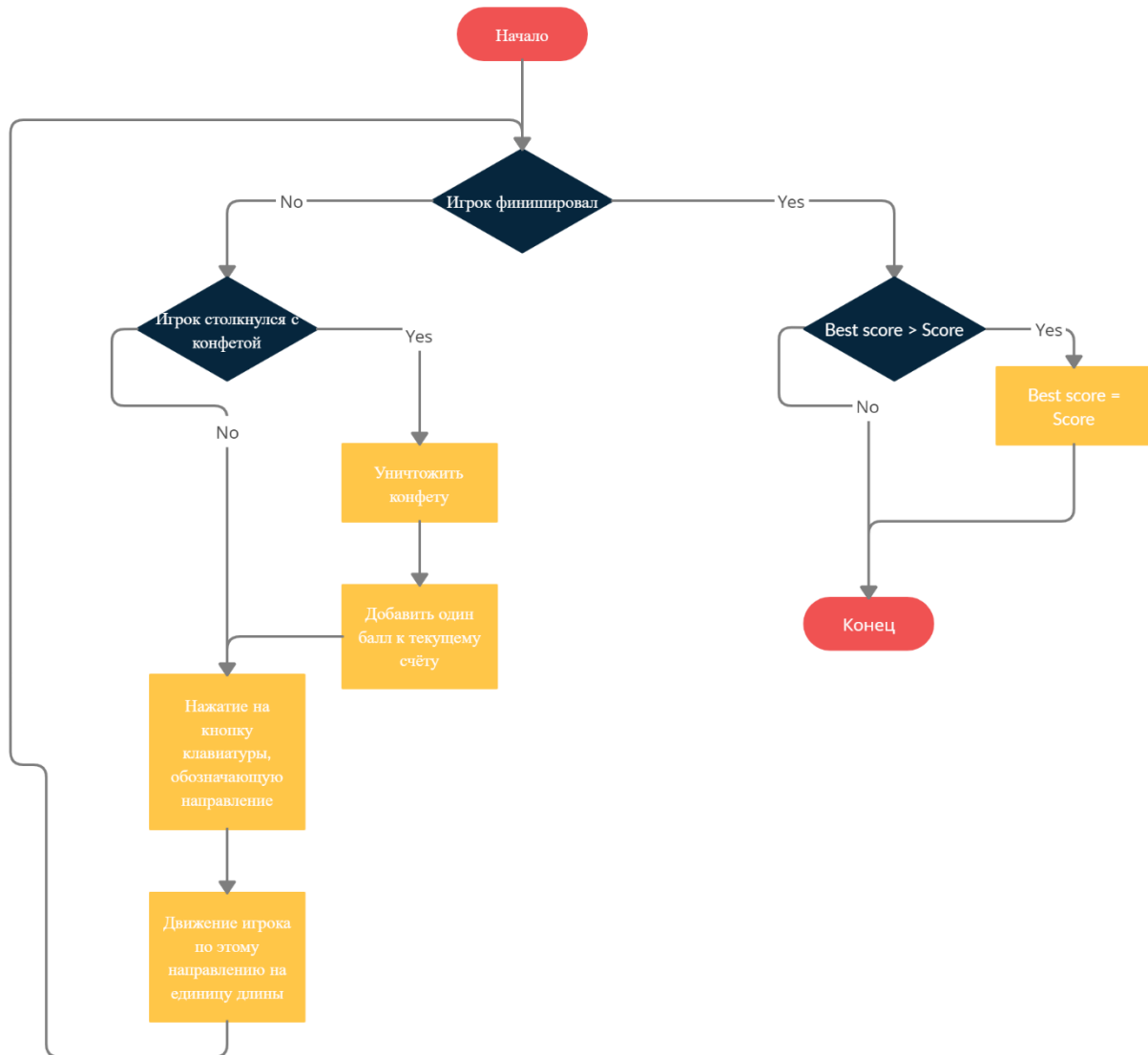
- Начальное значение счёта, я беру его равным нулю, но при желании можно поменять
Тип этой переменной целочисленный, так как нельзя получать не целое число очко; ограничений нет
- Начальное значение лучшего счёта, я беру равным 451. Так как максимальное количество собранных конфет может равняться 450, то 450 наихудший счёт, ведь цель игры набрать наименьшее возможное количество конфет. Тогда 451 хуже самого неудачного результата и тогда в любом случае при начале новой игры человек сможет получить свой лучший счёт.
Тип этой переменной целочисленный, так как счёт может выражаться только целым числом; диапазон от суммы начального счёта и 54, так как это наименьший результат и нет смысла делать лучший счёт таким, чтобы его не возможно было достичь, до любого целого числа
Но если человек уже играл в эту игру ранее и закрыл её, то он может поменять свой лучший результат на тот, что был раньше и пытаться побить именно этот результат.
- Скорость персонажа, у меня равняется 10, но любой игрок может изменить её и сделать шарик быстрее или медленнее.
Тип этой переменной float, так как значение скорости не обязательно должно быть целым числом; нельзя делать скорость равную 0, так как тогда шарик не сможет двигаться, делать скорость меньше 0 можно но не желательно, так как шарик будет двигаться противоположно заданному направлению (нажимаем вправо, двигается влево), не нужно делать слишком большую по модулю скорость, потому что будет не удобно играть

Выходными данными являются

- Счёт игрока – он считается прибавлением одного очка за каждую съеденную шариком конфетку. Выводится после финиширования.
Тип переменной целочисленный, так как невозможно собрать только часть балла; диапазон зависит от того, какое значение было взято в начале, но если считать, что первичный счёт равнялся 0, то минимальное (лучшее) значение 54, а максимальное (худшее) 450
- Лучший счёт игрока – он считается после финиширования, если игрок набрал меньше очков, чем в прошлой игре, то его лучший счёт меняется на текущий счёт, если игрок справился хуже, чем в прошлый раз, то лучший счёт остаётся неизменным.

Тип переменной целочисленный, так как не возможно собрать только часть балла; диапазон зависит от того, какое значение было взято в начале, но если считать, что первичный счёт равнялся 0, то минимальное (лучшее) значение 54, а максимальное (худшее) 450

✓ Математическая модель



Вектор движения игрока определяется вектором движения по каждой из осей. Положение игрока изменяется в определённую сторону на столько, на сколько изменилось положение игрока по этой оси.

Направление движения по осям считается исходя из того, какие кнопки клавиатуры, отвечающие за движение, были нажаты.

$$\text{Vector} = (\text{Vector.x} * \text{Speed}, \text{Vector.y}, \text{Vector.z} * \text{Speed}) * \text{Time}$$

Я обозначила за:

Vector – вектор итогового движения игрока

Vector.x - вектор движения по оси x

Vector.y – вектор движения по оси y (поэтой оси шар не движется)

Vector.z – вектор движения по оси z

Speed – скорость игрока

Time – время изменения кадра, чтобы движение было плавным

При столкновении с монеткой, к текущему счёту прибавляется один бал и монетка уничтожается, её нельзя будет собрать второй раз

Формула при столкновении:

Score = Score + 1;

Score – текущий счёт

Анализ используемой структуры данных

Все входные данные будут храниться в отдельных переменных.

Безусловно необходимо хранить данные о скорости движения шарика, иначе игрок не сможет двигаться и поиграть будет невозможно

также необходимо хранить данные о текущем и лучшем счёте, но они будут изменяться в течение игры.

Без текущего счёта в игре не будет смысла, тогда нужно будет просто бесцельно проходить лабиринт и тогда в этой игре будет на много меньше развивающих функций.

А лучший счёт делает игру интереснее, побуждает человека проходить игру несколько раз и стремиться к наилучшему результату.

Выходные данные, это и есть изменённые входные данные (счёт и лучший счёт) они необходимы в игре для наличия смысла в ней и заинтересованности игроков.

Выходные данные, получаем из переменных, в которых хранятся входные данные

Выбор метода решения

Я реализовывала игру на платформе Unity, программируя на языке C#.

- Для начала я спроектировала карту лабиринта и воссоздала её с помощью 3d объектов в Unity. То есть сделала пол, разместила стены, сделала зону для финиша и старта.
- Затем я настроила основную камеру, а после вторую. Сделала так, чтобы между ними можно было переключаться. Таким образом наиболее удобная для игры камера показывает вид сверху, а другая позволяет оглядеться вокруг и понять куда же нужно идти.
- Я настроила свет разместив его над всем лабиринтом.
- После я создала самого персонажа – нашего игрока Шонди. Запрограммировала его движение, чтобы им можно было управлять стрелочками клавиатуры и буквами WASD.
- Далее я создала конфетки, которые Шонди должен есть, расставила их по всему лабиринту, через 1 условную клеточку. И написала скрипт, благодаря которому Шонди смог их подбирать.
- Наладила счёт съеденных конфет
- Далее я решила сделать начальную страницу игры – меню. Для этого создала новую сцену, нарисовала фон и добавила кнопки для перехода далее и на сцену с правилами игры.

- Создала сцену с правилами игры; нарисовала правила и добавила в игру.
- Сделала финишную зону, и объявила её триггером для появления финальной сцены, итого счёта, лучшего счёта и кнопки перезагрузки.
- Запрограммировала как будет подсчитываться лучший счёт и сохраняться во время игры

Комментированный листинг

- Движение шарика

Для реализации движения шарика были использованы три переменные, обозначающие направление движения шарика.

Значение по оси x, получается при нажатии стрелок(букв) со значениями право/лево – “Horizontal”

Значение по оси y, получается при нажатии стрелок(букв) со значением верх/низ (в игре считается как вперёд/ назад) - “Vertical”

Чтобы движение было плавным, итоговый вектор умножаю на Time.deltaTime – даёт время для прохождения этого кадра.

```
private void CharacterMove()
{
    moveVector = Vector3.zero;
    moveVector.x = Input.GetAxis("Horizontal") *
    speedMove;
    moveVector.z = Input.GetAxis("Vertical") *
    speedMove;

    moveVector.y = gravityForce;
    ch_controller.Move(moveVector * Time.deltaTime);
}
```

- Собираение конфет и начисление очков

Если шарик сталкивается с конфетой, то конфета уничтожается (Destroy(other.gameObject))

Затем начисляется 1 балл (score++)

```
void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("PickUp"))
    {
        Destroy(other.gameObject);
        score++;
        score2 = score;
    }
}
```

- Смена сцен

Смена сцен происходит благодаря функции LoadScene. Эта функция загружает новую сцену с названием, которое написано в скобках.

Скрипт кусочек из которого представлен ниже, я помещала на кнопки.
Таким образом при нажатии на кнопку загружается новая сцена.

```
public class SceneGame : MonoBehaviour
{
    public void ChangeSceneGame()
    {
        SceneManager.LoadScene("Game");
    }
    public void ChangeSceneRulles()
    {
        SceneManager.LoadScene("Rulles");
    }
    public void ChangeSceneStart()
    {
        SceneManager.LoadScene("Start");
    }
}
```

- **Финальная картинка со счётом и лучшим счётом появляется из-за функции SetActive**

В начале игры картинка финиширования не видна, так как она находится в неактивном состоянии - SetActive(false)

При срабатывании триггера – это происходит, когда шарик попадает в зону с финишем, состояние картинки меняется на активное – SetActive(true) и мы её видим.

Аналогично происходит и с некоторыми другими объектами.

```
private void Start()
{
    FinishPage.SetActive(false);
    StartPage.SetActive(true);
    Camera.SetActive(true);
    Camera2.SetActive(true);
    FinishCamera.SetActive(false);
}

void OnTriggerEnter(Collider col)
{
    if(col.tag == "Player")
    {
        FinishPage.SetActive(true);
        StartPage.SetActive(false);
        Camera.SetActive(false);
        Camera2.SetActive(false);
        FinishCamera.SetActive(true);
    }
}
```

- Конфетки поворачиваются при использовании функции transform.rotation. Она меняет уже не положение объекта, как transform.position, а угол поворота. Чтобы конфетки вращались равномерно, использована функция Time.deltaTime. Quaternion.Euler – изменяет вращение по каждой оси в градусах

```
void Update()
```

```
{
    transform.rotation =
        Quaternion.Euler(transform.rotation.eulerAngles +
            rotation * Time.deltaTime);
}
```

- Переключение между двумя камерами происходит при нажатии на кнопку клавиатуры с английской буквой «Е». Программа проверяет какой ключ был подан и если это код буквы Е, то программа выключает камеру, которая осуществляла видимость сейчас и включает другую.

```
void Update()
{
    if (Input.GetKeyDown(KeyCode.E))
    {
        TheCamera.enabled = !TheCamera.enabled;
        Cam.enabled = !Cam.enabled;
    }
}
```

- Следование камеры за персонажем
Положение камеры задаётся transform.position исходя из положения шарика – нашей цели (target) и добавляет начальное расстояние между персонажем и камерой (offset)


```
void Start()
{
    offset = transform.position -
        target.transform.position;
}


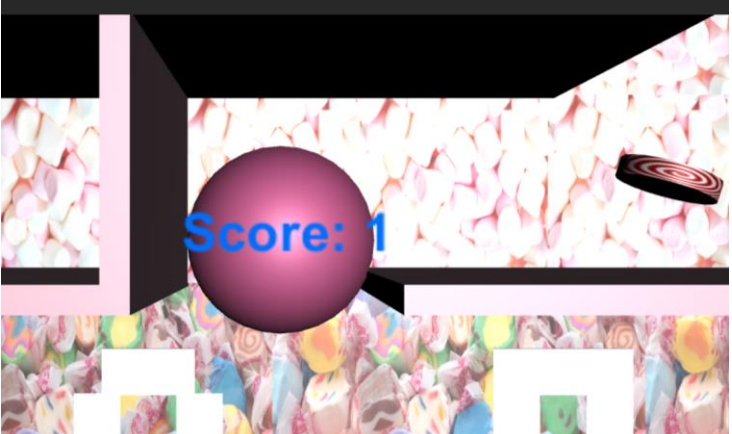

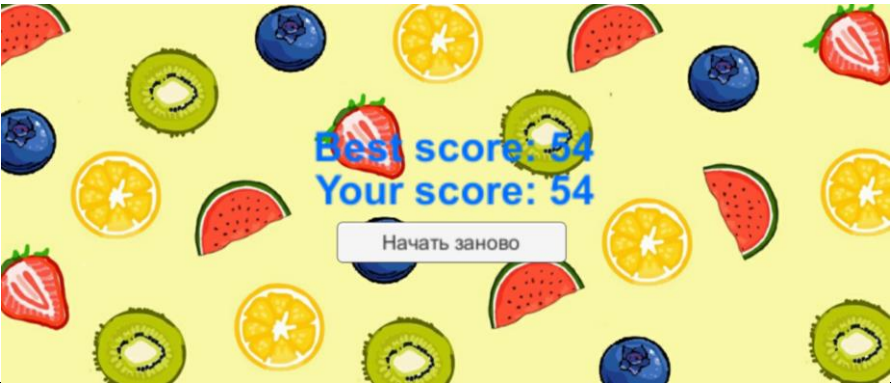
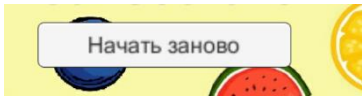
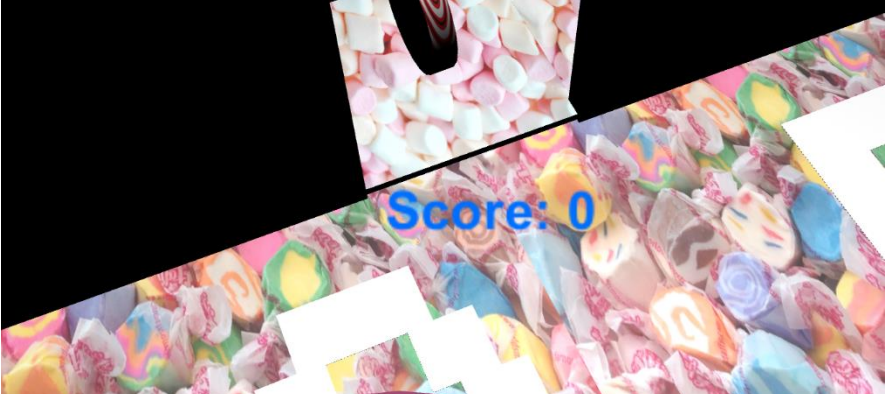
void Update()
{
    transform.position = target.transform.position +
        offset;
}
```

- Поворот камеры осуществляется благодаря передачи её позиции положение мышки на экране, то есть вектор направления мыши – это вектор поворота камеры.

```
void Update()
{
    lastMouse = Input.mousePosition - lastMouse;
    lastMouse = new Vector3(-lastMouse.y * camSens,
        lastMouse.x * camSens, 0);
    lastMouse = new Vector3(transform.eulerAngles.x +
        lastMouse.x, transform.eulerAngles.y + lastMouse.y,
        0);
    transform.eulerAngles = lastMouse;
    lastMouse = Input.mousePosition;
}
```

Пример работы программы

<p>На входе</p> <p>Нажимаем на кнопку с надписью правила</p> 	<p>Результат</p> <p>Переходим на сцену с правилами игры</p> 
<p>Нажимаем на кнопку «МЕНЮ» на сцене с правилами</p> 	<p>Возвращаемся на сцену с меню</p> 
<p>Нажимаем на кнопку «НАЧАТЬ»</p> 	<p>Начинается игра</p> 
<p>Нажимаем на кнопку «Е» на клавиатуре</p>	<p>Переключаем камеру</p> 
<p>Нажимаем на стрелочки/WASD</p>	<p>Двигаемся в соответствующие стороны</p>

<p>Собираем конфетку</p> 	<p>Получаем +1 балл к счёту</p> 
<p>Доходим до финиша</p> 	<p>Появляется итоговый счёт, лучший счёт, кнопка «начать заново»</p> 
<p>Нажимаем на кнопку «начать заново»</p> 	<p>Попадаем в начало игры</p> 

Анализ правильности решения

Программа работает верно, так как, например при столкновении с монеткой, она удаляется, а к счёту прибавляется 1 балл.

При нажатии соответствующих кнопок шарик движется в правильную сторону. При нажатии на кнопки открываются именно те сцены, о которых там было написано. В финале показывается тот счёт, который действительно был набран во время игры.