

ОТЧЕТ

о проектной работе

по теме: Memory Maze. Развитие памяти

по дисциплине: Проектный практикум

Команда: 10’cent Team

Екатеринбург

2021

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ](#_heading=h.80qsbw4hrhc9) 2

[ВВЕДЕНИЕ](#_heading=h.w2n8lehe8jtn) 3

[КОМАНДА](#_heading=h.sfsjy2qlf8d5) 4

[ЦЕЛЕВАЯ АУДИТОРИЯ](#_heading=h.nqu6zfpaoy7m) 5

[КАЛЕНДАРНЫЙ ПЛАН ПРОЕКТА](#_heading=h.8gjcnnntam5y) 6

[ОПРЕДЕЛЕНИЕ ПРОБЛЕМЫ](#_heading=h.91b1cqhof801) 8

[ПОДХОДЫ К РЕШЕНИЮ ПРОБЛЕМЫ](#_heading=h.lafjjfo6lpvm) 10

[АНАЛИЗ АНАЛОГОВ](#_heading=h.8xyxdw99hhrb) 11

[ТРЕБОВАНИЯ К ПРОДУКТУ И К MVP](#_heading=h.9qr86yx17hxl) 12

[СТЕК ДЛЯ РАЗРАБОТКИ](#_heading=h.gg2nhcsp2if9) 13

[ПРОТОТИПИРОВАНИЕ](#_heading=h.evd5guesunms) 14

[РАЗРАБОТКА СИСТЕМЫ](#_heading=h.8r9igkba9zw5) 20

[ЗАКЛЮЧЕНИЕ](#_heading=h.v0ftrvwc9w16) 21

[СПИСОК ЛИТЕРАТУРЫ](#_heading=h.whfekzig3osi) 22

[ПРИЛОЖЕНИЕ А](#_heading=h.r6rdzze2ox8p) 23

# ВВЕДЕНИЕ

Часто случается, что нужно запомнить некоторый объем информации за достаточно короткий срок и с минимальными тратами. Например, учебные материалы, контакты людей, места и адреса, маршруты, разновидности растений и многое другое.

Исходя из данной проблемы, были придуманы определенные методы для более эффективного запоминания.

Методов запоминания, так называемых мнемотехник, существует достаточно большое количество, всех их можно подразделять по типу информации, которую нужно запомнить. Самые распространенные методы – Ассоциации, Буквенный код, Рифмы, Созвучия, и, наконец, Дворец памяти, который является темой данного проекта.

В открытом доступе не так много образовательных игр, целью которых служит научить людей какой-либо методике запоминания информации. Перед нашей командой встала задача - знакомство с процессом развития памяти путем визуализации: создания в своём воображении пространства, в котором можно хранить огромное количество информации.

Игрок сможет развить память, познакомится с мнемоническим методом - Дворец памяти. Поймет принцип его работы и научится с ним работать в реальной жизни на примере игры.

Нашей командой была поставлена конкретная цель: создать игру, которая познакомит игрока с мнемоническим методом “Дворец памяти”, покажет принцип его работы и научит пользоваться им в реальной жизни.

Для достижения поставленной цели был выявлен ряд задач:

-Изучить методы запоминания информации, выбрать мнемотехнику для дальнейшей разработки.

-Описать ситуацию под выбранную мнемотехнику

-Выявить требования для обучения пользователя

-Составить календарный план проекта

-Установить макет уровней

-Разработать MVP

# 

# КОМАНДА

* Гафаров Владислав Ришатович РИ-100013 – Дизайнер
* Егоров Владислав Витальевич РИ-100013 – Программист
* Недосекина Александра Олеговна РИ-100015 – Тимлид
* Павленко Алексей Игоревич РИ-100002 – Программист
* Чуркин Кирилл Алексеевич РИ-100013 – Тестировщик

# ЦЕЛЕВАЯ АУДИТОРИЯ

- Люди начиная со школьного возраста, заинтересованные в развитии памяти и интересующиеся мнемотехникой.

- Нашей игрой заинтересуются люди, желающие познакомиться с мнемоническим приемом “Дворец Памяти”, люди, которым приходится запоминать большие объемы данных.

# КАЛЕНДАРНЫЙ ПЛАН ПРОЕКТА

# 

**Название проекта**: Memory Palace - игра, нацеленная на обучение пользователя мнемоническому методу «Дворец памяти»

**Руководитель проекта**: Недосекина Александра Олеговна

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **№** | **Название** | **Ответственный** | **Длительность** | **Дата начала** | **Временные рамки проекта** | | | |
| 1 нед | 2 нед | 3 нед | 4 нед |
| ***Анализ*** | | | | | | | | |
| *1.1* | *Определение проблемы* | Недосекина А.О | 3 дня | 8.04.21 |  |  |  |  |
| *1.2* | *Выявление целевой аудитории* | Егоров В.В | 3 дня | 8.04.21 |  |  |  |  |
| *1.3* | *Конкретизация проблемы* | Недосекина А.О | 3 дня | 8.04.21 |  |  |  |  |
| *1.4* | *Подходы к решению проблемы* | Недосекина А.О | 3 дня | 8.04.21 |  |  |  |  |
| *1.5* | *Анализ аналогов* | Чуркин К.А | 3 дня | 8.04.21 |  |  |  |  |
| *1.6* | *Определение платформы и стека для продукта* | Гафаров В.Р | 3 дня | 8.04.21 |  |  |  |  |
| *1.7* | *Формулирование требований к MVP продукта* | Павленко А.И | 3 дня | 8.04.21 |  |  |  |  |
| *1.8* | *Определение платформы и стека для MVP* | Павленко А.И | 3 дня | 8.04.21 |  |  |  |  |
| *1.9* | *Формулировка цели* | Егоров В.В | 3 дня | 8.04.21 |  |  |  |  |
| *1.10* | *Формулирование требований к продукту* | Гафаров В.Р | 3 дня | 8.04.21 |  |  |  |  |
| *1.11* | *Определение задач* | Чуркин К.А | 3 дня | 8.04.21 |  |  |  |  |
|  | *…* |  |  |  |  |  |  |  |
| ***Проектирование*** | | | | | | | | |
| *2.1* | *Архитектура системы (компоненты, модули системы)* | Чуркин К.А | 1 неделя | 16.04.21 |  |  |  |  |
| *2.2* | *Разработка сценариев использования системы* | Егоров В.В | 1 неделя | 16.04.21 |  |  |  |  |
| *2.3* | *Прототипы интерфейсов* | Гафаров В.Р | 1 неделя | 18.04.21 |  |  |  |  |
| *2.4* | *Дизайн-макеты* | Гафаров В.Р | 1 неделя | 18.04.21 |  |  |  |  |
|  | *…* |  |  |  |  |  |  |  |
| ***Разработка*** | | | | | | | | |
| *3.1* | *Написание кода* | Павленко  А.И Егоров В.В | 3 недели | 2.05.21 |  |  |  |  |
| *3.2* | *Тестирование приложения* | Чуркин К.А | 1 неделя | 16.05.21 |  |  |  |  |
|  | *...* |  |  |  |  |  |  |  |
| ***Внедрение*** | | | | | | | | |
| *4.1* | *Оформление MVP* | Павленко А.И  Егоров В.В | 1 неделя | 19.05.21 |  |  |  |  |
| *4.2* | *Внедрение MVP* | Павленко А.И  Егоров В.В | 1 неделя | 19.05.21 |  |  |  |  |
| *4.3* | *Написание отчета* | Недосекина А.О | 1 неделя | 20.05.21 |  |  |  |  |
| *4.4* | *Оформление презентации* | Гафаров В.Р | 1 неделя | 24.05.21 |  |  |  |  |
|  | *…* |  |  |  |  |  |  |  |
|  | *Защита проекта* | Недосекина А.О |  | 07.06 - 15.06 |  |  |  |  |

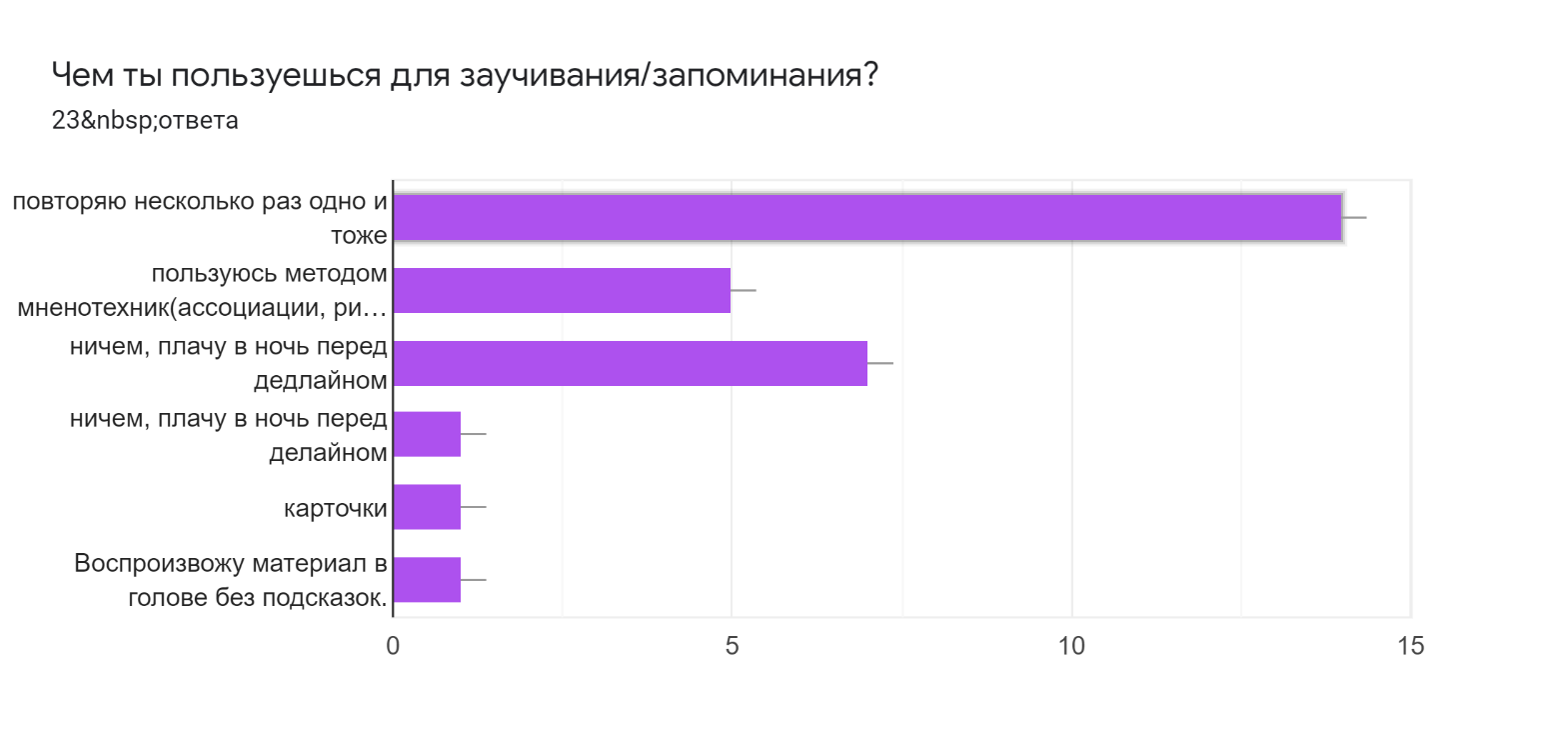
# ОПРЕДЕЛЕНИЕ ПРОБЛЕМЫ

# 

Наша команда задалась вопросом: «как обычный человек - наш пользователь- вообще запоминает информацию и с помощью чего?»

Исходя из вопроса, было принято решение провести опрос на базе «Google формы»

Сначала мы узнали как человек запоминает информацию, результаты можно просмотреть на диаграмме :



Как можно увидеть, чаще всего пользователь просто повторяет материал до тех пор, пока не сможет свободно воспроизвести его без подсказок.

Также отслеживается, что метод мнемотехник не пользуется большим спросом.

Далее мы спросили «как бы вы стали изучать метод мнемотехники?»

Результат можно увидеть на диаграмме:



По результатам, мы пришли к выводу, что самый популярный вариант, учиться – играя, то есть познакомиться с методом мнемотехники в виде образовательной игры.

Также отслеживается вариант – изучение за счет проcмотра обучающих видео на платформе YouTube. Это бесспорно самый прогрессивный вариант.

Наименее популярный вариант – читать о методе в книгах или интернете. Не самый удобный способ, зато самый достоверный.

Итак, наше решение должно представлять собой образовательную игру, так как это самый популярный и наиболее удобный способ обучения методу мнемотехники для пользователя.

# ПОДХОДЫ К РЕШЕНИЮ ПРОБЛЕМЫ

Традиционный:

* обнаружение проблемы
* постановка задач для обучения мнемотехники
* корректировка и оценка результата обучения
* закрепление результата и вывод

# АНАЛИЗ АНАЛОГОВ

# 

Проводя анализ конкурентов, нашей командой был сделан вывод, что разрабатываемый продукт не имеет ни прямых, ни косвенных аналогов.

В то же время у игры есть конкуренты, которые имеют как схожие, так и отличные стороны.

* «Mnemocon - развитие Памяти и Внимания Тренажер ума» мобильная игра, затрагивает несколько методов мнемотехники сразу.
* «Мнемонист - тренировка памяти и мозга» мобильная игра, затрагивает несколько мнемотехник.
* «Мнемотехники» веб-сервис компании «4brain», курс из 5 недель интенсивного обучения мнемотехнике Цицерон (Дворец памяти)

Также существует огромное количество дидактических игр, суть которых заключается в запоминании изображений на карточках, восстановлении их хронологии и связи друг с другом.

# ТРЕБОВАНИЯ К ПРОДУКТУ И К MVP

* Поддержка пк, (mobile)
* Оптимизация
* Интуитивно понятный интерфейс
* Обучить мнемотехнике - Дворец памяти
* Единый визуальный стиль
* Лабиринт

# 

# 

# СТЕК ДЛЯ РАЗРАБОТКИ

Нашей командой был выбран игровой движок – Unity, так как это быстрая и гибкая платформа для разработки на очень многие платформы.

Язык программирования - с# был выбран исходя из логики того, что данный язык изучается на предмете “Программирование”

Платформа – ПК , так как задумывалось, что будет именно desktop игра, в дальнейшем возможна мобильная адаптация.

# ПРОТОТИПИРОВАНИЕ

Первоначальные требования : Наличие лабиринта, игровых уровней - комнат, вид 3-го лица

1-я переработка прототипа :

вид от 3 лица , реализовано игровое пространство - комната.

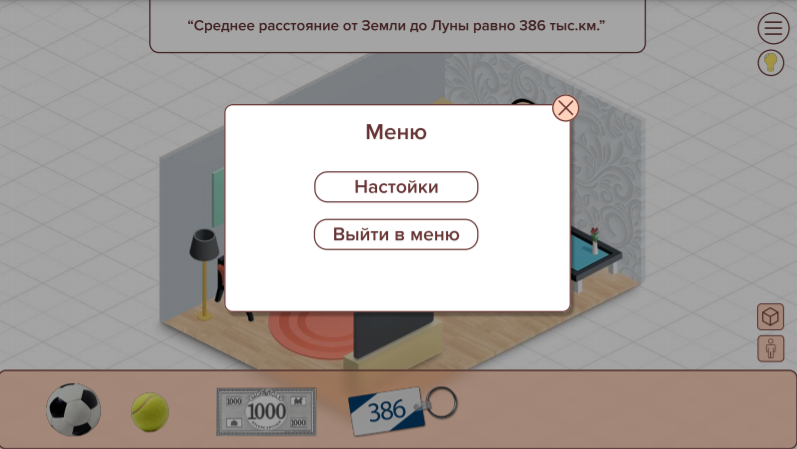
2-я переработка прототипа :

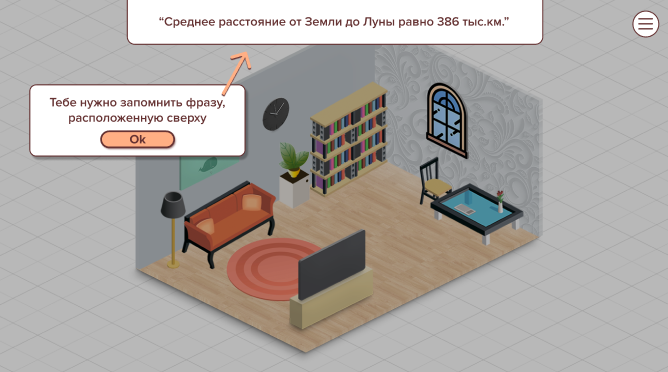
Сделано обучение, игровое меню, прототип 2-д лабиринта.

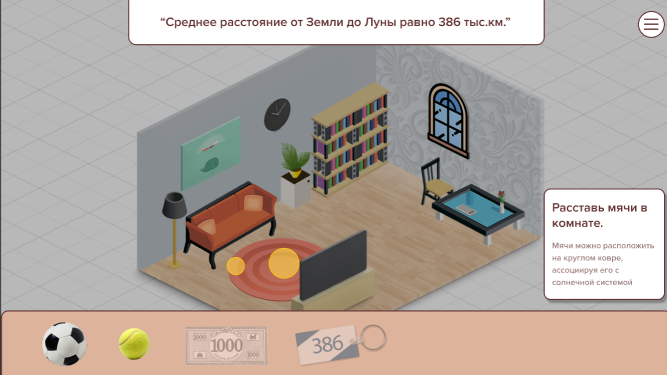
3-я переработка прототипа:

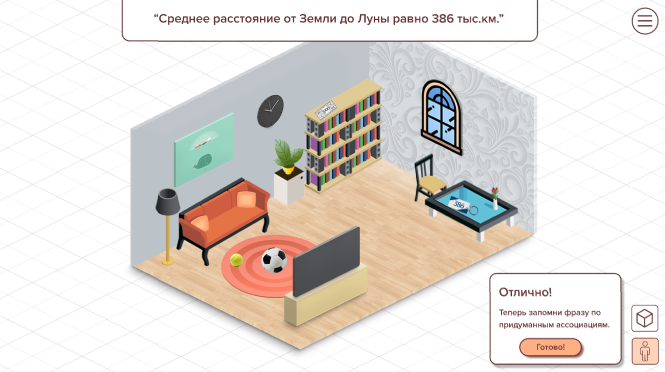
Небольшие улучшения интерфейса, конечный вариант лабиринта

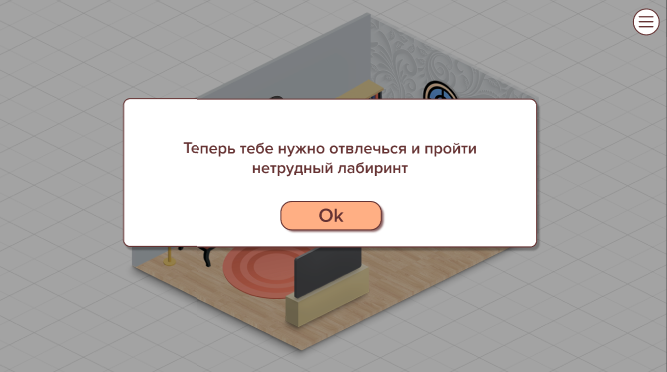


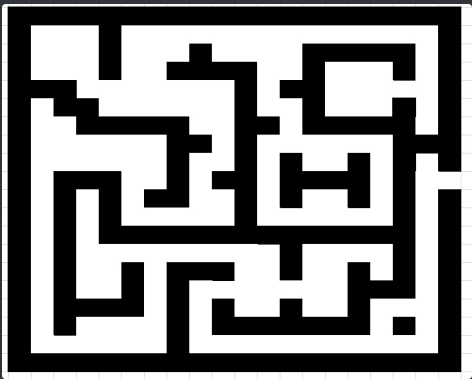


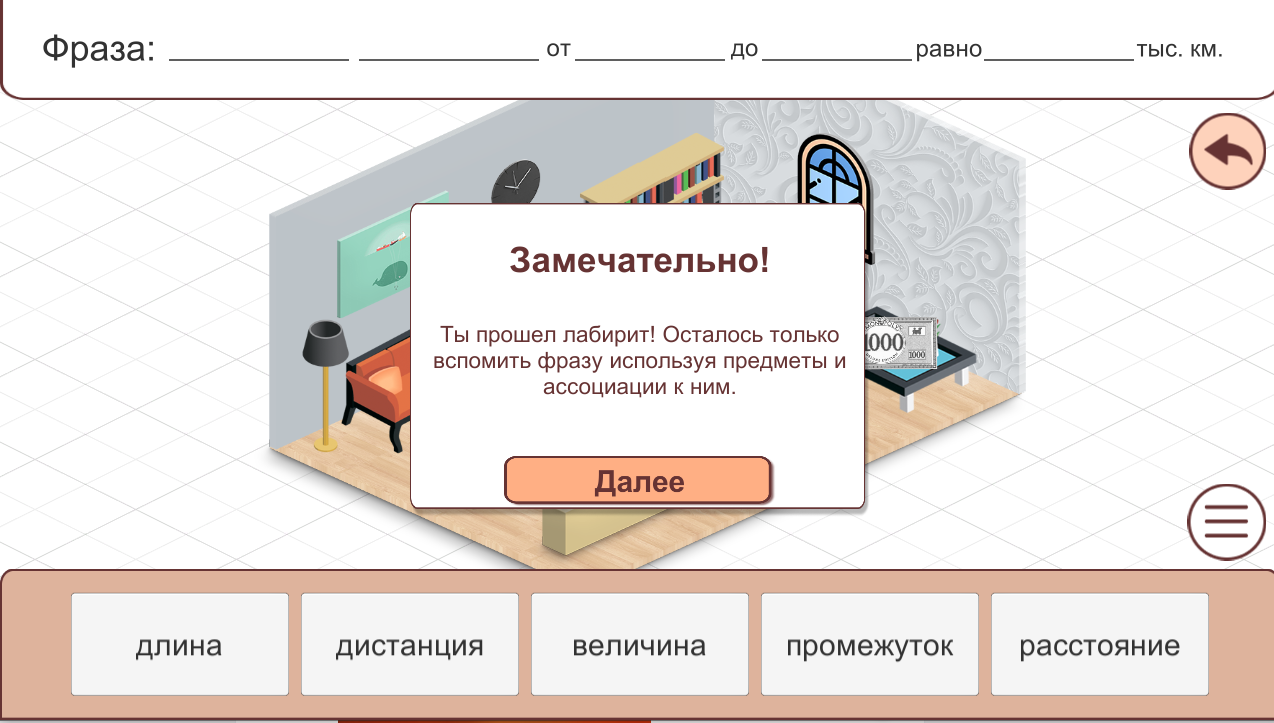




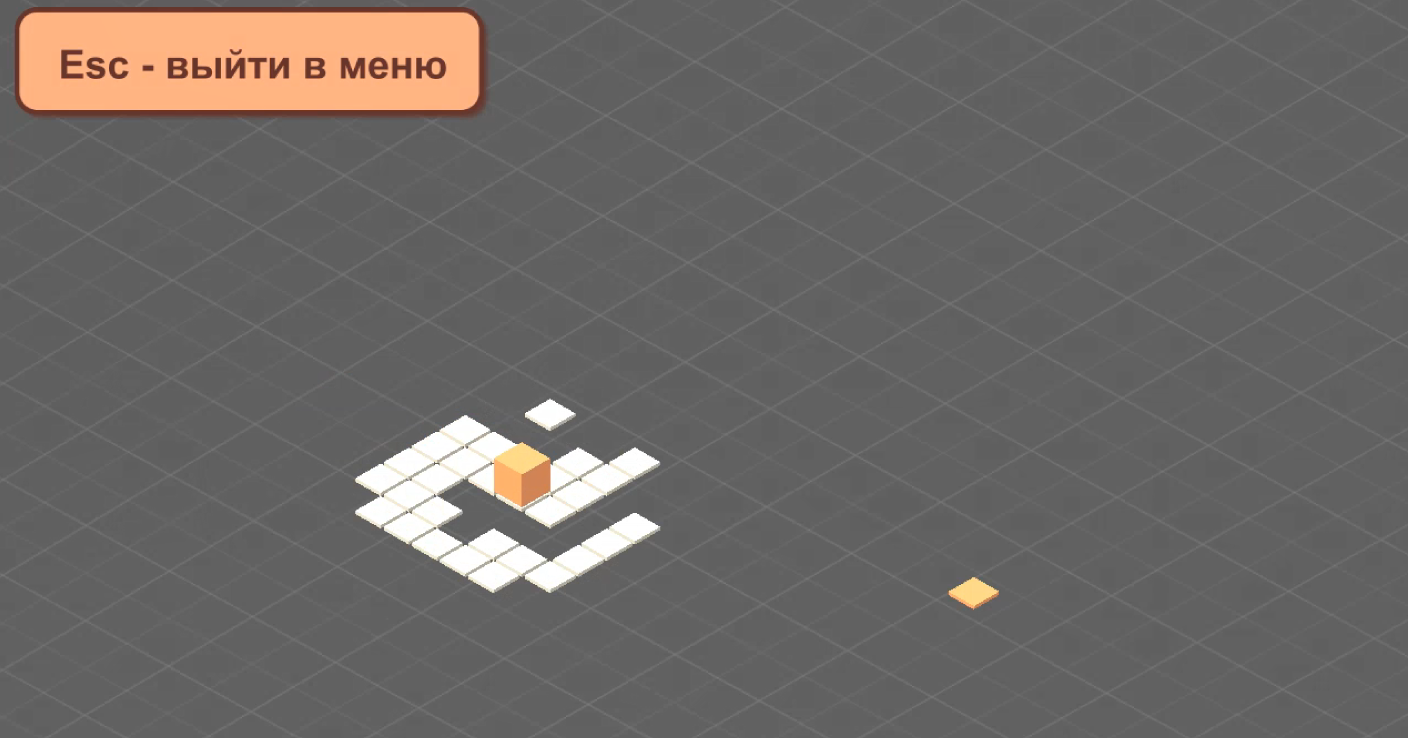












# РАЗРАБОТКА СИСТЕМЫ

Этапы игры:

— Обучение:

Игрок знакомится с интерфейсом, техниками запоминания и возможностями их применения на примере метода дворца. Обучающий уровень, на котором игроку уже предоставлены готовые примеры использования метода, игрок только использует их для запоминания.

— Повторение:

Небольшой заранее подготовленный уровень, в котором игроку предлагается запомнить легкую фразу. Доступны наводящие подсказки при необходимости.

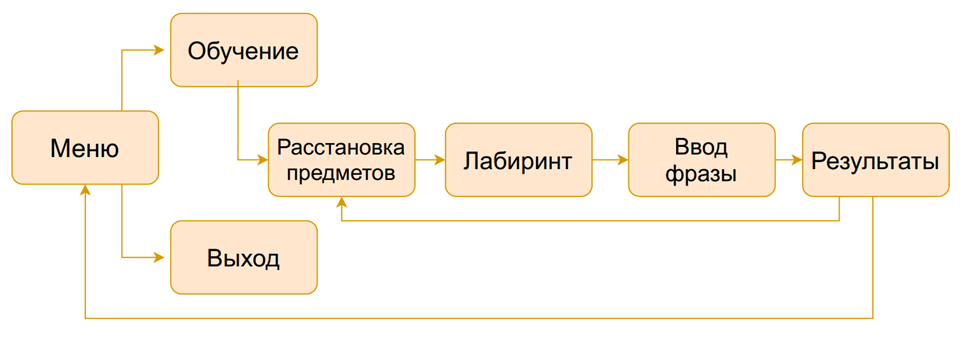
— Закрепление:

Сложность фраз для запоминания повышается. Игрок строит ассоциативные связи самостоятельно, подсказки недоступны.

— Испытание (проверка знаний):

Появляются предметы, которые могут сбить с толку игрока. Появляется ограничение по времени на прохождение уровня и количество неправильных попыток.

Все листинги кода приведены в ПРИЛОЖЕНИИ А



# ЗАКЛЮЧЕНИЕ

Главная задача нашей команды заключалась в знакомстве пользователя с мнемоническим методом «Дворец памяти».

Мы разработали MVP(Minimum viable product ) - создали «пространство» в котором можно хранить огромное количество информации, встроили мини-игру – лабиринт.

В процессе создания особых трудностей не возникало, так как мы старались выполнять поставленные задачи последовательно, консультировались с кураторами, на каждой итерации ставились задачи с ответственными, детальным описанием и определенными сроками исполнения.

Дизайнер создал макеты игровых сцен, воспроизвел «Дворец» на макете в Figma.

Для непосредственно разработки игры и оживления макетов в Unity 3D потребовалось изучать с нуля стек разработки , читать официальную документацию , форумы , а также смотреть туториалы на платформе Youtube.

У нас получилась полноценная обучающая игра. Метод дворца заключается в свободном размещении представленных предметов по комнате, запоминании фразы за счет расположения этих предметов и воспроизведении ее же после прохождения лабиринта, который заставляет мозг переключиться от запоминания на мышление.

В процессе создания игры были трудности с программной частью. Проект необходимо было выгрузить на электронный ресурс icho.io для того, чтобы можно было играть в браузере без загрузки на компьютер. Для устранения этой проблемы проект нуждался в адаптивной верстке, о чем мы узнали очень поздно и не пошли по этому пути в силу временных ограничений и отсутствия знаний в данной области.

В дальнейшем будут добавлены новые локации , реализован режим добавления собственных фраз и также будет пополнен багаж доступных предметов для ассоциаций.

Также планируется мобильная адаптация игры на платформы IOS и Android.

# СПИСОК ЛИТЕРАТУРЫ

1. Мнемотехники [Электронный ресурс] https://4brain.ru/memory/mnemotehniki.php
2. Stackoverflow [Электронный ресурс]  
   https://ru.stackoverflow.com/
3. Документация Unity 3D [Электронный ресурс]  
   https://docs.unity3d.com/ru/530/Manual/
4. Туториал по созданию игр [Электронный ресурс]  
   https://www.youtube.com/watch?v=pwZpJzpE2lQ
5. Cyberforum [Электронный ресурс]  
   https://www.cyberforum.ru/unity/thread2252739.html
6. Coderoad [Электронный ресурс]  
   <https://coderoad.ru/>

# ПРИЛОЖЕНИЕ А

**Start.cs**

public class Start : MonoBehaviour

{

public void OnClick()

{

SceneManager.LoadScene("room"); //загрузить комнату

}

}

**Exit.cs**

public class Exit : MonoBehaviour

{

public void OnClick()

{

Application.Quit(); //закрыть приложение

}

}

**ToMenu.cs**

public class ToMenu : MonoBehaviour

{

public void OnClick()

{

SceneManager.LoadScene("MainMenu");//загрузка сцены главного меню

}

}

**DragItem.cs**

public class DragItem : MonoBehaviour, IPointerDownHandler, IDragHandler, IEndDragHandler

{

[SerializeField] private Canvas canvas;

private RectTransform rectTransform;

public GameObject Board;

public GameObject Room;

private bool isOnLowerLayer;

private bool isHoverOnBoard;

private bool isOnBoard;

public float halfHeight;

public float halfWidth;

private bool isInScreen = true;

private const int UpBorder = 100;

private const int SideBorder = 150;

private void Start()

{

Canvas.ForceUpdateCanvases();//обновляем все канвы

rectTransform = GetComponent<RectTransform>();//кэшируем компонент объекта

halfHeight = (canvas.GetComponent<RectTransform>().rect.height - 50) / 2;

halfWidth = (canvas.GetComponent<RectTransform>().rect.width - 50) / 2; //вычисляем максимальные и минимальные координаты канвы

}

public void Update()

{

if (isHoverOnBoard && !isOnBoard) //если объект над доской и не прикреплен к ней, то прикрепить

{

transform.SetParent(Board.transform);

transform.SetSiblingIndex(1);

isOnBoard = true;

return;

}

if (isHoverOnBoard) return;

isOnBoard = false;

transform.SetParent(Room.transform); //иначе прикрепить к комнате

}

public void OnPointerDown(PointerEventData eventData)//вызывается при нажатии

{

if (eventData.button == PointerEventData.InputButton.Right) //если нажата правая кнопка мыши, то сменить слой объекта в комнате

{

var siblingsCount = transform.parent.childCount;

if (isOnLowerLayer)

transform.SetSiblingIndex(siblingsCount - 1);

else

transform.SetSiblingIndex(2);

isOnLowerLayer = !isOnLowerLayer;

}

}

public void OnDrag(PointerEventData eventData)//вызывается при перетаскивании

{

if (isInScreen && eventData.button == PointerEventData.InputButton.Left) //если курсор в границах окна и нажата левая кнопка мыши

{

var nextPos = rectTransform.anchoredPosition + eventData.delta / canvas.scaleFactor; //вычисляем следующую позицию

if (nextPos.x < -halfWidth + SideBorder || nextPos.x > halfWidth - SideBorder ||

nextPos.y < -halfHeight || nextPos.y > halfHeight - UpBorder) //проверяем что она выходит за границы окна

{

isInScreen = false; //курсор больше не в пределах экрана

}

nextPos.x = Mathf.Clamp(nextPos.x, -halfWidth + SideBorder, halfWidth - SideBorder);

nextPos.y = Mathf.Clamp(nextPos.y, -halfHeight, halfHeight - UpBorder); //кореектируем перемещение чтобы удержать объект в границах

rectTransform.anchoredPosition = nextPos; //переносим объект

}

}

public void OnEndDrag(PointerEventData eventData) //вызывается при окончании перетаскивания

{

isInScreen = true; //курсор снова в границе экрана на начало перетаскивания, т.к. все объекты в границах экрана

}

private void OnTriggerEnter2D(Collider2D other) //вызывается при начале коллизии объектов

{

if (other.gameObject.CompareTag("board")) //если другой объект – доска, то предмет находится над ней

{

isHoverOnBoard = true;

}

}

private void OnTriggerExit2D(Collider2D other) //вызывается по окончании коллизии объектов

{

if (other.gameObject.CompareTag("board")) //если другой объект – доска, то предмет более не находится над ней

{

isHoverOnBoard = false;

}

}

}

**SceneChanger.cs**

public class SceneChanger : MonoBehaviour

{

public List<GameObject> toEnable;

public GameObject Tutorial;

public GameObject FadeoutObj;

public GameObject FadeinObj;

public void OnClick()

{

gameObject.GetComponent<Button>().interactable = false; //делаем кнопку неактивной, чтобы избежать лишних нажатий

StartCoroutine(ChangeToMaze());

}

private IEnumerator ChangeToMaze()

{

FadeoutObj.SetActive(true); //начинаем затемнение

yield return new WaitForSeconds(2);

Tutorial.SetActive(false); //выключаем последний туториал

foreach (var gameObject in toEnable)

{

gameObject.SetActive(true); //включаем все нужные объекты

}

SceneManager.LoadScene("maze", LoadSceneMode.Additive); //начинаем загрузку лабиринта

yield return new WaitWhile(() => !SceneManager.GetSceneByName("maze").isLoaded);

yield return new WaitWhile(() => SceneManager.GetSceneByName("maze").isLoaded);//ждем пока игрок не выйдет из лабиринта

gameObject.GetComponent<Button>().interactable = true;//снова активируем кнопку для дальнейшего использования

FadeinObj.SetActive(true); //начинаем высветление

FadeoutObj.SetActive(false);//убираем затемнение

yield return new WaitForSeconds(2);

FadeinObj.SetActive(false);//заканчиваем высветление

}

}

**MazeMenu.cs**

public class MazeMenu : MonoBehaviour

{

public GameObject Menu;

public GameObject Tutorial;

void Update()

{

if (Input.GetKeyDown(KeyCode.Escape) && !Menu.activeSelf) //если меню не открыто и нажат escape

{

Menu.SetActive(true); //открыть меню

}

else if (Menu.activeSelf) //если меню активно

{

if (Input.GetKeyDown(KeyCode.N) || Input.GetKeyDown(KeyCode.Escape)) //если нажаты escape или “N”

{

Menu.SetActive(false);//закрыть меню

}

else if (Input.GetKeyDown(KeyCode.Y)) //если нажата “Y”

{

Cursor.lockState = CursorLockMode.None; //разблокировать курсор

SceneManager.LoadScene("MainMenu"); //перейти в главное меню

}

}

else if (Tutorial.activeSelf) //если открыт туториал

{

if (Input.GetKeyDown(KeyCode.Return)) //если нажата Enter

{

Tutorial.SetActive(false); //закрыть туториал

}

}

}

}

**Player.cs**

using System;

using System.Drawing;

using System.Threading.Tasks;

using UnityEngine;

using UnityEngine.SceneManagement;

public class Player : MonoBehaviour

{

[SerializeField] private KeyCode keyForward = KeyCode.W;

[SerializeField] private KeyCode keyBackward = KeyCode.S;

[SerializeField] private KeyCode keyLeft = KeyCode.A;

[SerializeField] private KeyCode keyRight = KeyCode.D;

[SerializeField] private Vector3 moveDirection = Vector3.forward;

[SerializeField] private Vector3 moveSide = Vector3.right;

public GameObject Platform;

public GameObject ExitPlatform;

public GameObject FadeoutObj;

private const float playerHeight = 0.65f;

public Map Map = new Map(

@"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*###\*##############\*

\*###\*###\*####\*\*\*\*\*#\*

\*###\*##\*\*\*\*##\*###\*#\*

\*\*\*#######\*#\*\*#####\*

\*#\*\*######\*##\*###\*#\*

\*##\*\*\*\*\*##\*\*#\*\*\*\*\*#\*

\*######\*\*#\*######\*\*\*

\*######\*##\*#\*##\*#\*#\*

\*#\*\*\*##\*#\*\*#\*\*\*\*#\*##

\*#\*#\*#\*\*##\*#\*##\*#\*#\*

\*#\*#\*#####\*######\*#\*

\*#\*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*#\*

\*#\*#########\*####\*#\*

\*#\*##\*#\*\*\*##\*##\*#\*#\*

\*#\*##\*#\*#######\*\*\*#\*

\*#\*\*\*\*#\*#\*##\*##\*###\*

\*#\*####\*#\*\*\*\*\*\*\*#\*#\*

\*######\*###########\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*", new Point(2,2), new Point(19,9)); //создание карты с указаниями точек входа и выхода

private void Start()

{

SceneManager.SetActiveScene(SceneManager.GetSceneByName("maze")); //установить текущую сцену активной

Cursor.lockState = CursorLockMode.Locked; //заблокировать курсор

GetComponent<Transform>().position = new Vector3(Map.start.X, playerHeight, Map.start.Y); //переместить игрока в точку входа

for (var x = 0; x < Map.map.GetLength(0); x++)

{

for (var z = 0; z < Map.map.GetLength(0); z++) //проходимя по матрице карты

{

if (Map.map[x,z]) //если есть платформа

{

if (Map.exit.X == x && Map.exit.Y == z) //если платформа на точке выхода

{

Instantiate(ExitPlatform, new Vector3(x, 0, z), new Quaternion()); //создать платформу выхода

continue;

}

var instantiate = Instantiate(Platform, new Vector3(x, 0, z), new Quaternion()); //иначе создать обычную платформу

var plat = instantiate.transform.Find("Platform").gameObject; //извлечь саму платформу из только что созданного объекта

var playerCoord = GetComponent<Transform>().position; // получить координаты игрока

var platformCoord = instantiate.GetComponent<Transform>().position;

if (!((platformCoord - playerCoord).magnitude > 3)) continue; //если плафторма ближе 3 единиц то продолжить

plat.GetComponent<Renderer>().enabled = false; //иначе отключить ее видимость

}

}

}

}

private void Update()

{

if (Input.GetKeyDown(keyForward)) //здесь и далее при нажатии определенной кнопки переместить на соответсвующее направление если возможно

{

if (Map.Check(GetComponent<Transform>().position + moveDirection)) //проверка на возможность перемещения

{

GetComponent<Transform>().position += moveDirection; //перемещение

Map.IfExitLeaveScene(GetComponent<Transform>().position, FadeoutObj); //покинуть лабиринт если игрок в точке выхода

}

}

if (Input.GetKeyDown(keyBackward))

{

if (Map.Check(GetComponent<Transform>().position - moveDirection))

{

GetComponent<Transform>().position -= moveDirection;

Map.IfExitLeaveScene(GetComponent<Transform>().position, FadeoutObj);

}

}

if (Input.GetKeyDown(keyRight))

{

if (Map.Check(GetComponent<Transform>().position + moveSide))

{

GetComponent<Transform>().position += moveSide;

Map.IfExitLeaveScene(GetComponent<Transform>().position, FadeoutObj);

}

}

if (Input.GetKeyDown(keyLeft))

{

if (Map.Check(GetComponent<Transform>().position - moveSide))

{

GetComponent<Transform>().position -= moveSide;

Map.IfExitLeaveScene(GetComponent<Transform>().position, FadeoutObj);

}

}

}

}

public class Map

{

public readonly bool[,] map; //карты

public readonly Point start; //точка входа

public readonly Point exit; //точка выхода

public Map(string map, Point start, Point exit)

{

this.map = CreateMap(map); //создать карту из ее матрицы

this.start = start;

this.exit = exit;

if (!this.map[exit.X, exit.Y])

throw new ArgumentException(); //если выход карты пуст, то кинуть исключение

}

public bool Check(Vector3 vector) //проеврка на корректность движения

{

return vector.x >= 0 && vector.x < map.GetLength(0) &&

vector.z >= 0 && vector.z < map.GetLength(1) &&

map[(int) vector.x, (int) vector.z];

}

private static bool[,] CreateMap(string map) //создание карты

{

var lines = map.Split(new[] {'\r', '\n'}, StringSplitOptions.RemoveEmptyEntries); //разбиение матрицы на строки

var result = new bool[lines[0].Length, lines.Length];

for (var x = 0; x < lines[0].Length; x++)

for (var y = 0; y < lines.Length; y++)

result[x, y] = lines[y][x] == '#'; //если в матрице стоит “#”, то создать платформу

return result;

}

public async void IfExitLeaveScene(Vector3 position, GameObject fadeoutObj)

{

if (new Point((int) position.x, (int) position.z) == exit) //если грок на точке выхода

{

fadeoutObj.SetActive(true); //начать затемнение

await Wait();

Cursor.lockState = CursorLockMode.None; //разблокировать кусрор

SceneManager.UnloadSceneAsync(SceneManager.GetActiveScene()); //выгрузить текущую сцену

}

}

private async Task Wait()

{

await Task.Delay(TimeSpan.FromSeconds(2));

}

}

**Platform.cs**

public class Platform : MonoBehaviour

{

private void OnTriggerEnter(Collider other) //вызывается при начале коллизии объектов

{

GetComponent<Renderer>().enabled = true; //выключить видимость плафтормы

}

private void OnTriggerExit(Collider other1) //вызывается по окончанию коллизии объектов

{

GetComponent<Renderer>().enabled = false; //отключить видимость платформы

}

}

**ButtonPhrase.cs**

public class ButtonPhrase : MonoBehaviour

{

public GameObject WordGroups;

public GameObject GroupToEnable;

public GameObject Words;

public Color StandartColor;

public Color PressColor;

public void OnClick()

{

foreach (Transform word in Words.transform) //пройтись по всем пропускам

{

word.GetChild(1).GetComponent<Image>().color = StandartColor; //отключить выделение пропуска

}

gameObject.transform.GetChild(1).GetComponent<Image>().color = PressColor; //включить выделение текущего пропуска

foreach (Transform wordGroup in WordGroups.transform) //пройтись по всем группам слов для вставки

{

if (wordGroup.gameObject == GroupToEnable) //включить нужную группу

{

wordGroup.gameObject.SetActive(true);

continue;

}

wordGroup.gameObject.SetActive(false); //отключить остальные группы

}

}

}

**ChangeWord.cs**

public class ChangeWord : MonoBehaviour

{

public GameObject ButtonToChange;

public void OnClick()

{

var textToChange = ButtonToChange.transform.GetChild(0).gameObject; //выделить текст для изменения

var text = transform.GetChild(0).gameObject; //выделить текст данной кнопки

textToChange.GetComponent<Text>().text = text.GetComponent<Text>().text; //изменить текст

textToChange.GetComponent<Text>().color = Color.black; //установить черный цвет текста

}

}

**CheckPhrase.cs**

public class CheckPhrase : MonoBehaviour

{

public List<string> ExpectedWords;

public GameObject WordsGroup;

public Sprite GoldenStar;

public GameObject LeftStar;

public GameObject RightStar;

public GameObject CentralStar;

public GameObject Screen;

public GameObject ErrorText;

public GameObject CheerPhrase;

public GameObject EmptyWordError;

public GameObject LevelCompletedText;

private int ErrorsCount;

public void CheckWords()

{

var index = 0;

var correctCount = 0;

foreach (Transform word in WordsGroup.transform) //пройтись по всем пропускам

{

var text = word.transform.GetChild(0).gameObject.GetComponent<Text>();

if (text.text == "") //если пропуск не заполнен

{

ErrorsCount = 0; //сбросить кол-во ошибок

EmptyWordError.SetActive(true); //показать предупреждение

return;

}

}

foreach (Transform word in WordsGroup.transform) //пройтись по всем пропускам

{

var text = word.transform.GetChild(0).gameObject.GetComponent<Text>();

if (ExpectedWords[index] == text.text) //если пропуск заполнен верно

{

text.color = Color.black; //окрасить в черный

correctCount++; //посчитать его как корректный

}

else

{

text.color = new Color(0.527f, 0.108f, 0.108f); //иначе окрасить в красный

ErrorsCount++; //увеличить кол-во сделанных ошибок

}

index++;

}

if (correctCount == ExpectedWords.Count) //если все пропуски заполнены верно

{

Screen.SetActive(true); //показать результаты

var cheerText = CheerPhrase.GetComponent<Text>(); //извлечь текст резултатов

if (ErrorsCount < 5) //здесь и далее при определенном пороге ошибок выводится фраза и зажигается какое-то кол-во звезд

{

LevelCompletedText.GetComponent<Text>().text = "Уровень пройден!";

cheerText.text = "Старайся лучше!";

cheerText.fontSize = 80; //разер фразы

LeftStar.GetComponent<Image>().sprite = GoldenStar; //зажигание звезды

}

if (ErrorsCount < 3)

{

RightStar.GetComponent<Image>().sprite = GoldenStar;

cheerText.text = "Неплохо";

cheerText.fontSize = 114;

}

if (ErrorsCount == 0)

{

CentralStar.GetComponent<Image>().sprite = GoldenStar;

cheerText.text = "Молодец!";

cheerText.fontSize = 114;

}

ErrorText.GetComponent<Text>().text = $"Ошибок: {ErrorsCount}"; //вывод кол-ва ошибок

ErrorsCount = 0; //сбросс кол-ва ошибок

}

}

}