**Курсов проект № 7**

**ООП с С# .NET**

**Snake Game**

Име: Николай Стоянов Натов

ФН: 62460

***1. Представяне на проекта***

Проектът представлява имплементация на аркадната игра *Snake* като са добавени някои допълнителни и различни функционалности и правила от тези в оригиналната игра. Основната цел на играта е, използвайки мишката, играчат да навигира змията така, че да може да изяде възможно най-много плодове. При всяко изяждане на плод, тялото на змията расте и играчът получава точки. Плодовете се появяват през определено време на случайни места от полето като различните видове плодове носят различен брой точки. Главата на змията не трябва да докосва друга част от тялото си или препятствията по игровия терен тъй като това ще доведе до край на играта. Играта има три нива на трудност – лесно, средно, трудно. Всяко ниво на трудност определя броя на препятствията на терена и интервала от време, в който се появяват плодовете на полето.

За разработката на проекта са използвани езикът за програмиране С# и платформата .NET, като графичният интерфейс е изграден чрез WPF.

***2. Архитектура на приложението***

Приложението е разработено на база MVVM(Model-View-Viewmodel) моделът. Това ни предоставя по-голяма гъвкавост тъй като логиката на играта е напълно независима от графичният интерфейс, което ни дава възможност за лесни промени върху него без да трябва да се пренаписва отново цялата игра. Така изградено приложението може лесно да се пренесе на други платформи или операционни системи като Android, iOS и др., които не използват графичен интерфейс написан на WPF.

Основните класове необходими за всяка игра(клас за помощни математически функции, клас за игрови обект, клас за игрови свят или пък клас отговярящ за заредените ресурси на играта като нива или текстури) за изнесени в отделен проект, който представлява така нареченият „игрови двигател“. Този проект е от тип Dynamic-link library и тази библиотека се включва към същинското приложение, което представлява цялата логика на играта, както и нейният интерфейс. Специфичната игра, която създаваме може да създава произволни игрови обекти, които наследяват базовият клас предоставен от библиотеката. Този модел предоставя преизползваем код за различните видове игри, както и лесно разширяване на функционалността на приложението.

Комуникацията между графичният интерфейс и обектите движещи логиката на играта се осъществява чрез клас наречен viewmodel, който предоставя цялата необходима за графичният интерфейс функционалност под формата на публични свойства и команди. Така WPF приложението използва като свой data context този viewmodel клас и чете от неговите свойства.

***3. Структури от данни***

Основната използвана структура от данни е ObservableCollection, която пази информация за всеки обект съществуващ в игровият свят. Тази структура от данни бива обхождана всеки кадър от играта проверявайки за колизии между обектите.

***4. Алгоритми***

* Движение на змията – движението на змията се осъществява като се използват координатите на мишката върху прозорецът на приложението. Обектът, който представлява главата на змията следи курсорът на мишката, а всеки следващ елемент от змията следи своят „родител“. Поската на движение на змията се изчислява като се построи векторът между двете точки(позицията на мишката и позицията на главата на змията). Полученият вектор на посоката се нормализира и умножава по скоростта на змията, след което се добавя към позицията на главата на змията. Ъгълът на ротация на главата на смията се изчислява като се намери ъгълът между векторът на посоката и абцисната ос. Останалите части от тялото на змията използват същият алгоритъм като единствената разлика е, че вместо да следват позицията на мишката следват предходният елемент на змията.
* Колизия – тъй като всеки елемент е представен като правоъгълник, за да проверим за колизии трябва да проверим дали точка от правоъгълникът представляващ главата на змията не се намира в друг правоъгълник представляващ обект от игралният свят (като например препятствие). За тази цел се създава bounding box за всеки правоъгълник като това става с умножение на всеки от върховете на правоъгълника с матрицата на трансформация на игровия обект. Получените точки след трансформацията представляват върховете на търсеният bounding box. Умножението с тази матрица е необходимо за да знаем позицията на правоъгълника и неговата ротация в пространството. Ако не го направим ще получим неточен bounding box. Накрая се използва вградена функция на С#, която проверява дали два правоъгълника се пресичат.

Забележка: понеже визуално не всеки обект е точен правоъгълник (например змията е текстура с прозрачни пиксели на някои места, която е използвана за оцветяване на площта на правоъгълник) е възможно тестът за колизия да даде положителен резултат, въпреки че в действителност не изглежда така.

* Инициализация на игровият свят – играта използва текстура с ниска резолюция, в която цветът на всеки пиксел представлява различен вид обект от света. Тази текстура се зарежда в паметта и се четат стойностите на четирите канала(RGBA) на всеки пиксел и в зависимост от това се добавят различни обекти на определени места от света.

**References:**

* **MSDN C# Documentation –**

[**https://docs.microsoft.com/en-us/dotnet/csharp/**](https://docs.microsoft.com/en-us/dotnet/csharp/)

* **Unity Engine move function –**

[**https://answers.unity.com/questions/414829/any-one-know-maths-behind-this-movetowards-functio.html**](https://answers.unity.com/questions/414829/any-one-know-maths-behind-this-movetowards-functio.html)

* **MVVM Design Pattern –**

[**https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm**](https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm)