Документация на проект **NFS**

изготвен от Мартин Добринов Иванов, 5. група, факултетен № 81602

1. Анализ на задачата и подход за решаването й

Проектът, който избрах аз, представлява опростена конзолна версия на играта Need For Speed. Основата цел на проекта е да се реализират състезания между играчи – реални и виртуални. Играта е направена така, че приятели могат да се съберат вкъщи и да играят заедно помежду си или срещу компютри. Играта разполага с магазин, от който реалните състезатели могат да си купуват по-мощни автомобили и така шансът им за победа се увеличава. Играта разполага с два вида автомобили – обикновени и с нитро. Играе се на рундове, в които се състоят двубоите. След всеки рунд играчите имат възможност да прекратят играта, както и да изиграят още един рунд. Играчите могат да се състезават само по двойки, като победителят се определя спрямо уменията и колата на всеки от тях. В края на всеки рунд се изготвя ранглиста, в която се подреждат играчите по техните умения от всички изиграни досега рундове. Така участниците могат да следят как се представят по време на двубоите.

Избрах да реализирам играта чрез набор от класове, т.е. използвайки парадигмата на Обектно-ориентираното програмиране. По услоние играта трябва да е компактно прибрана в един клас – това е именно класа **Nfs**, в който се прави връзката между всички останали класове и се извършва самата игра. В реализацията на играта има три основни момента – участниците, колите и магазинът. Ще опиша всеки от тях накратко.

Преди това, тъй като общият брой на играчите, както и броят на тези от тях, които са реални, се въвежда от потребителя, избрах да използвам динамичен едномерен масив от произволен тип елементи, като реализирах такъв чрез класа **dynamicArray**. Още повече, ние не знаем колко коли общо ще си купи един реален играч за цялата игра и там също ни е нужен този масив, за да съхраняваме всичките заупени от него коли. Класът е шаблонен, защото се налага използването му както за играчи (**Competitor**), така и за коли (**Car**). По-подробната му реализация е описана във втората точка.

Първият важен клас е абстрактният клас **Competitor** и неговите наследници **RealCompetitor** – представляващ реалните играчи, т.е тези, които сами решават как да играят, и **VirtualCompetitor** – виртуалните играчи (“компютри”), действията на които се определят на произволен принцип от компютъра. Избрах тук да има наследяване, тъй като реалните и виртуалните състезатели имат не малко общи характеристки, но притежават и различия – например виртуалните играчи нямат нужда от пари, тъй като те не могат да си купуват нови коли. Вместо да създавам еднакви полета и за двата класа, предпочетох да създам по-общ клас **Competitor**, който да съдържа тези общи характеристики, и наследеници, които да съдържат собствените си характеристики, които се отнасят само до тях. Още повече, избрах класът **Competitor** да бъде абстрактен, тъй като той обобщава общата информация за двата вида играчи, т.е. той съдържа техните общи характеристики. Обекти от този клас няма да бъдат създавани, защото всеки от играчите е специализиран – или е реален, или е виртуален.

Следващият създаден от мен клас е **Car**. Негови наследници са класовете **NormalCar** и **NitroCar**. Всяка кола е или обикновена, или нитро кола. И двата вида притежават общи характеристики като сериен номер, цена и точки, които носи на играча при победа, но имат и различия като например нитро колата получава бонус точки на всяко трето състезание (с цел презареждане на нитрото). И двата наследника притежават общи черти, но всеки от тях си има собствени характеристики, типични само за него. Затова обобщих еднаквите характеристики на двата типа коли в класа **Car** (базовия), а специфиичните за всеки от тях – в класа за съответната кола (наследниците).

Третият основен клас е **Shop**, който представлява магазинът за коли. Със създаването на маазина той се зарежда с коли, измежду които играчите могат да си избират и купуват. На всеки рунд всеки от играчите има опция да отиде до магазина и да си купи по-хубава кола. Основната цел на магазина е да предлага и продава на играчите по-мощни коли, за да могат те да печелят повече състезания. По условие, колите са предварително дефинирани в програмата, затова със създаването на обект от този тип, той автоматично се зарежда с конкретни, сходни по характеристики, но разнообразни коли (прави се в конструктора). Конкретната реализация е описана в точка 2.

Най-важният клас е **Nfs**, защото той осъществява връзката между всички компоненти на играта. Вътре в него се генерират играчите, колите, осъществява се изборът на действия от всеки ирач, както и двубоите между тях, извежда се ранглистата след всеки рунд. В него е реализирана основната функционалност на играта. В следващата точка е описано подробно как точно се случва това.

В реализацията на проекта главно се работи с указатели към обектите от съответния тип, повечето създадени масиви са от указатели към обекти. Вместо да създавам отделни масиви от обекти от тип класовете, които са наследници на някои от основните класове, аз реших да създам масив от указатели към базовия клас и съответно да дефинирам функциите, които ще се ползват в наследниците по специфичен начин, като виртуални. Деструкторите на базовите класове са виртуални, за да може при разрушаване на обект да се изтриват както наследените член-данни, така и собствените за наследника. Избрах този начин за реализация, защото така се опростява кодът. Както и защото този начин е по-добър при решение за разширяване на проекта и създаване на по-сложни йерархии.

Играта се стартира през външна функция. Избрах да е така, вместо с член-функцция на класа **Nfs**, защото при следващо надграждане на проекта в тази функция може да се добавят нови функционалности, например – по време на състезанието да има зрители, които да наблюдават и гласуват за участниците в състезанието – ще е нужно да се създаде обект от тип Spectators и това ще стане именно в тази външна функция.

1. Описание на класовете, техните член-данни и член-функции
   1. Класът **dynamicArray**

За нуждите на играта реших да използвам шаблонен клас за едномерен динамичен масив. Тъй като той използва динамична памет, за него е дефинирана голямата четворка – конструктор по подразбиране, деструктор, конструктор за присвояване и оператор за присвояване. Реших масивът да има капацитет, като при достигането му той се увеличава с константа. Чрез това се намалява постоянното освобождаване на заета памет от хийпа и заделяне на нова при добавянето на елементи вътре в масива. Предефинирах оператора [], за да мога по-лесно да достъпвам елементите на масива. Има функции за добавяне и премахване на елементи от масива, които се използват по време на играта. Например при разделението на играчите по двойки за състезанието.

* 1. Класовете **Competitor**, **RealCompetitor**, **VirtualCompetitor**

Общите характеристики за двата типа състезатели са ID-то им и уменията им. Затова те са обединени в класа **Competitor**, като добавих сетъри и гетъри за двете член-данни. Реалните състезатели могат да притежават множество от коли, защото могат да си купуват такива от магазина. Затова те имат член-данна, която е динамичен масив от указатели към коли. Виртуалните състезатели имат само една кола – тази, която им се даде на произволен принцип в началото на играта. Затова те имат член-данна, която е указател към една кола. Всеки състезател си има най-добра кола (определя се от мощността на колата), затова в класа **Competitor** създадох чисто виртуална член-функцията **virtual** **Car\* Competitor::getBestCar();**, която определя коя е най-добрата кола. Тя е чисто виртуална, защото начинът, по който се избира най-добра кола за съответния състезател е различен, не може да бъде обобщен еднозначно в базовия клас за състезателите, но и за двата типа играчи трябва да бъде избрана най-добрата от колите им. При реалните състезатели най-добрата кола се определя измежду всички, които играчът си е купил до този момент, докато при виртуалните най-добрата кола е тази, която е дадена в началото, те не могат да имат други коли изобщо. Това се извършва още при закупуването на нова кола. Тогава се проверява дали нейната мощност е по-голяма от мощността на вече определената за най-добра кола.

* 1. Класовете **Car**, **NormalCar**, **NitroCar**

Единствената разлика между обикновените коли и нитро колите е тази, че нитро колите получават допълнителни точки на всеки трети рунд от играта. Затова двата типа коли са разделени в два класа, в противен случай обикновените коли нямаше да използват по никакъв начин бонус точките и щяха да са излишни при тях. Дефинирани са конструктори и принт-функция за всеки от класовете, защото само те са ни нужни за реализацията на играта. Единственото място, където се създават коли, е при зареждането на магазина. След това всички коли на играчите се определят от тези в магазина.

* 1. Класът **Shop**

Според условието на проекта колите в магазина са предварително дефинирани. Затова съм ги реализирал чрез статичен масив от указатели към коли. За всеки елемент (те са указатели) от масива се създава кола в динамичната памет – 10 обикновени и 10 нитро коли със сходни мощности и цени. Разположени са в динамичната памет, защото ще бъдат използвани и от други класове и техните член-функции. Паметта се освобождава в деструктора на магазина. Направих и гетър за колите в магазина, за да е по-лесно достъпването им.

* 1. Класът **Nfs**

-член-данни

В този клас се осъществява същинската игра и връзката между играчите, колите, магазина. Тъй като броят на играчите се определя от потребителя, създадох динамичен масив от указатели към играчи. За да може всеки от играчите да си купува кола, добавих член-данна от тип магазин, която ще представлява връзката на играчите с магазина, и по-точно с колите в него. Тъй като на всеки играч му се принтира списък с играчите, срещу които може да се състезава, добавих два динамични масива **Array<RealCompetitor> isFreeRealEnemy** и **Array<RealCompetitor>isFreeVirtualEnemy** от булеви стойности, в които се записва кога даден играч – реален или виртуален, е избран вече за противник. Разпределението на играчите по двойки се записва в динамичния масив **Array<int>enemies;**.

- член-функции

**void Nfs::generateCompetitors();** – В началото всеки играч започва с произволна кола от наличните (т.е. тези от магазина). За да генерирам произволен избор на кола, използвам функцията **rand()** от библиотеката **cstdlib** и функцията **srand()** от библиотеката **ctime**, първо за определяне дали произволната кола да бъде обикновена или нитро и след това за определяне коя точно от първите две коли от съответния тип да получи играча. Тъй като те са сходни по мощност, условието да няма големи различия между играчите при стартирането на състезанието е изпълнено.

**void Nfs::chooseAction();** – На всеки рунд всеки играч има по два варианта – да си купи нова кола и да се състезава или направо да се състезава. Този избор се извършва именно от тази член-функция. Ако играчът избере да си купи кола, той „отива“ в магазина, купува си колата и след това избира противник за състезанието. Ако избере направо да се състезава, се принтира списък с играчите, срещу които може да се състезава, както и подходяща информация за всеки от тях. Направена е и валидация на входа.

**void Nfs::buyCar(Competitor\*);** – това е функцията, която осъществява връзката между играчите и магазина. На всеки играч му се принтира списък с наличните коли в магазина, мощността им и цената им. След това те избират какъв тип кола искат да си купят и ако имат пари, я купуват, след което тя се добавя към динамичния масив от коли на съответния реалнен състезател. След това на играча се принтира в конзолата списък за избор на противник.

**void Nfs::chooseEnemy();** – На всеки играч се принтира списък с играчите, срещу които може да се състезава. Чрез член-функцията **void Nfs::showFreeEnemies();** се принтират само тези, които все още не са били избрани. Играчът избира дали да се състезава срещу реален или виртуален играч, избира съответното ID на играча. След това ID-то на избиращия и на избрания се вкарват в масива **enemies**, откъдето по-късно ще се определят двубоите за състезанието.

**void Nfs::chooseRandomEnemy();** – Възможен сценарий е такъв, при който част от виртуалните състезатели останат неизбрани. Те са четен брой, защото иначе няма как да проведем състезанието. Заради това добавих тази фунция, която ги разпределя по двойки на произволен принцип. Създава се един временен масив **Array<int>tempFreeEnemies;**, в който се съхраняват ID-тата на тези от виртуалните играчи, който все още нямат противник, както и още един масив от булеви стойности **Array<int>usedCompetitors;**, в който ще се пази дали вече сме намерили противник на някои от свободните играчи. Сега, чрез **rand()** по произволен начин се избира едно от ID-тата в масива **tempFreeEnemies** и се добавя в **enemies**. Това се прави докато всички свободни играчи не бъдат вкарани в масива **enemies**. Тогава за да разберем как ще играят по двойки, вземаме всеки два съседни елемента от масива и те определят една двойка за игра.

**void Nfs::compete();** – Тук се извършва определянето на победителя от всяка от разпределените вече двойки. Сравняват се техните коефициенти, които се изчисляват по дадена в условието формула. Победителят печели пари и умение, което му помага при участие в следващите рундове. Във формулата за определяне на коефициента на всеки от състезателите участва доза късмет, т.е. на произволен принцип се генерира число между 0 и 20. За да се реализира това, използвах функцията **rand()**.

**void Nfs::ranking();** – След като се определят победителите, се принтира класацията до този момент. Има две отделни класации – за реалните за виртуалните състезатели. Реализирано е чрез сортиране на играчите по умението им до този момент.

1. Идеи за бъдещо подобрение
   1. Да се добавят нови коли в магазина, а не да са предварително дефинирани в началото на състезанието
   2. Освен коли да се продават и части за тези коли
   3. Играчите да могат да продават коли един на друг
   4. Да може да се добавя допълнително нитро по време на състезанието и по този начин играчът да има по-голям шанс за победа