|  |
| --- |
| Изображение выглядит как текст, коллекция картинок, фарфор  Автоматически созданное описание  МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-2 «Прикладные информационные технологии» |

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №1**по дисциплине  
«Технологии и методы программирования»

*(наименование дисциплины)*

Выполнил студент группы: БИСО-01-20 20Б0869 Ламзин А. С.

*(шифр учебной группы) (Фамилия И.О.)*

Проверил Лесько С. А.

*(Фамилия И.О.)*

Работа выполнена «\_\_»\_\_\_\_\_2023г.

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Москва 2022

1. Определить главных или первичных и второстепенных актеров:

Главными актерами являются класс Game, так как объединит в себе другие классы, и классы Player и Machine.

Второстепенными актерами являются классы Board, Dice, Square, PropertySquare, UtilitySquare и RailroadSquare. Они представляют собой компоненты игры для возможности играть.

1. Определить цели главных актеров по отношению к системе

Цель игрока и машины - перемещаться по квадратам на игровом поле, зарабатывать деньги, покупать и продавать недвижимость, утилиты и железнодорожные станции и в конечном итоге победить, т.е. остаться единственным игроком с деньгами.

Цель игры (Game) состоит в том, чтобы предоставить игрокам возможность играть в игру, контролировать правила игры, перемещение игроков, розыгрыш кубиков, покупку и продажу недвижимости и т.д.

1. Сформулировать основные варианты использования, которые специфицируют функциональные требования к системе

Игрок может Начать игру

Игрок может Бросить кубики

Игрок может Купить имущество

Игрок может Продать имущество

Игрок может Пройти на старт

Игрок может Закончить игру

Машина может Начать игру

Машина может Бросить кубики

Машина может Купить имущество

Машина может Продать имущество

Машина может Пройти на старт

Машина может Закончить игру

1. Упорядочить варианты использования по степени убывания риска их реализации
   1. Вариант "Игрок выбрасывает кубики"
   2. Вариант "Машина выбрасывает кубики
   3. Вариант "Игрок попадает на клетку и выполняет действие"
   4. Вариант "Машина попадает на клетку и выполняет действие"
   5. Вариант "Игрок покупает свойство"
   6. Вариант "Машина покупает свойство"
   7. Вариант "Игрок платит аренду за чужое свойство"
   8. Вариант "Машина платит аренду за чужое свойство"
2. Рассмотреть все базовые варианты использования в порядке убывания их степени риска
   1. Game.play(): Этот вариант использования относится к основной функциональности системы, не представляет большого риска
   2. Player.takeTurn(): относится к основной функциональности систем. Но различные аспекты игры, такие как перемещение по доске, управление имуществом, сбор арендной платы, могут повлиять на выполнение этого варианта использования.
   3. Square.landOn():также является частью основной функциональности игры. Существует возможность ошибок при обработке разных типов клеток.
   4. Player.pay() и Player.receive(): риск - несоблюдение правил игры, что может привести к ошибкам, неправильному расчету денег и проблемам в дальнейшем.
   5. Board: риск связан с тем, что при неправильном создании доски могут возникнуть ошибки, такие как бесконечные циклы, выход за пределы массивов и т.п.
   6. Dice.roll(): Этот вариант использования может быть реализован с наименьшим риском, поскольку он не требует сложных вычислений и зависит только от генерации случайных чисел.
3. Выделить участников, интересы, предусловия и постусловия выполнения выбранного варианта использования

Участники:

* Игрок
* Класс Game
* Класс Dice

Интересы:

* Игрок: получить случайное число от 1 до 6 для передвижения на поле
* Класс Game: обеспечить случайное число при броске кости и обновить позицию игрока
* Класс Dice: сгенерировать случайное число от 1 до 6

Предусловия:

* Игрок должен быть тем игроком, чей ход сейчас

Постусловия:

* Класс Game получает случайное число от класса Dice
* Класс Game обновляет позицию игрока на поле по полученному числу

1. Написать успешный сценарий реализации выбранного варианта использования

Игрок начинает свой ход и бросает кости

Класс Game вызывает метод roll() класса Dice для генерации случайного числа

Класс Dice генерирует случайное число от 1 до 6

Класс Game обновляет позицию текущего игрока на игровом поле в соответствии с полученным числом

Игрок перемещается на новую клетку и выполняет действие на ней

1. Определить исключения или неуспех в выполнении сценария варианта использования
2. Объект Game не создан
3. Объект Dice не создан
4. Кубик вылетел за пределы доски во время броска
5. Написать сценарии для всех исключений
6. Если объект Game не создан, то нажатие кнопки "Бросить кубики" не приведет к выполнению метода roll() объекта Dice.
7. Если объект Dice не создан, то метод roll() не будет выполнен и результат броска кубиков не будет получен.
8. Игрок бросает кубики; Кубик вылетает за пределы доски; Игра прерывается; Игрокам сообщается, что бросок неудачный и нужно перебросить кубики

Скриншоты выполненных кодов:

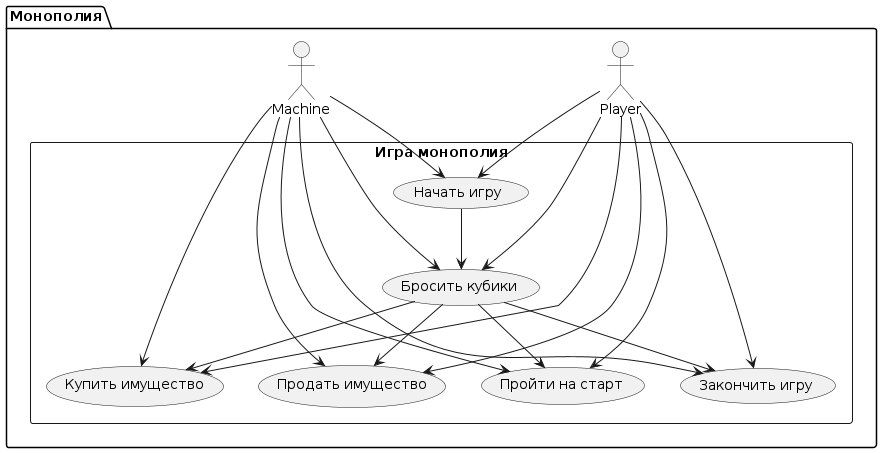


Рисунок 1 – диаграмма вариантов использования

Game.play(): метод, который запускает игру

Game.endGame(): метод, который завершает игру

Board.squares: поле, которое содержит массив квадратов на игровой доске

Square.landOn(player: Player): метод, который вызывается при приземлении игрока на ячейку

PropertySquare.landOn(player: Player): метод, который вызывается при приземлении игрока на ячейку недвижимости

UtilitySquare.landOn(player: Player): метод, который вызывается при приземлении игрока на ячейку утилит

RailroadSquare.landOn(player: Player): метод, который вызывается при приземлении игрока на ячейку железной дороги

Player.takeTurn(): метод, который позволяет игроку бросить кубики и переместиться на соответствующее количество ячеек

Player.pay(amount: int): метод, который уменьшает количество денег игрока на указанную сумму

Player.receive(amount: int): метод, который увеличивает количество денег игрока на указанную сумму

Player.cash: поле, которое содержит текущее количество денег игрока

Player.currentSquare: поле, которое содержит текущее положение игрока на доске

Player.properties: поле, которое содержит массив имущества игрока

Machine.takeTurn(): метод, который позволяет машине бросить кубики и переместиться на соответствующее количество ячеек

Machine.pay(amount: int): метод, который уменьшает количество денег машины на указанную сумму

Machine.receive(amount: int): метод, который увеличивает количество денег машины на указанную сумму

Machine.cash: поле, которое содержит текущее количество денег машины

Machine.currentSquare: поле, которое содержит текущее положение машины на доске

Machine.properties: поле, которое содержит массив имущества машины

Dice.roll(): метод, который генерирует случайное число, имитируя бросок кубиков

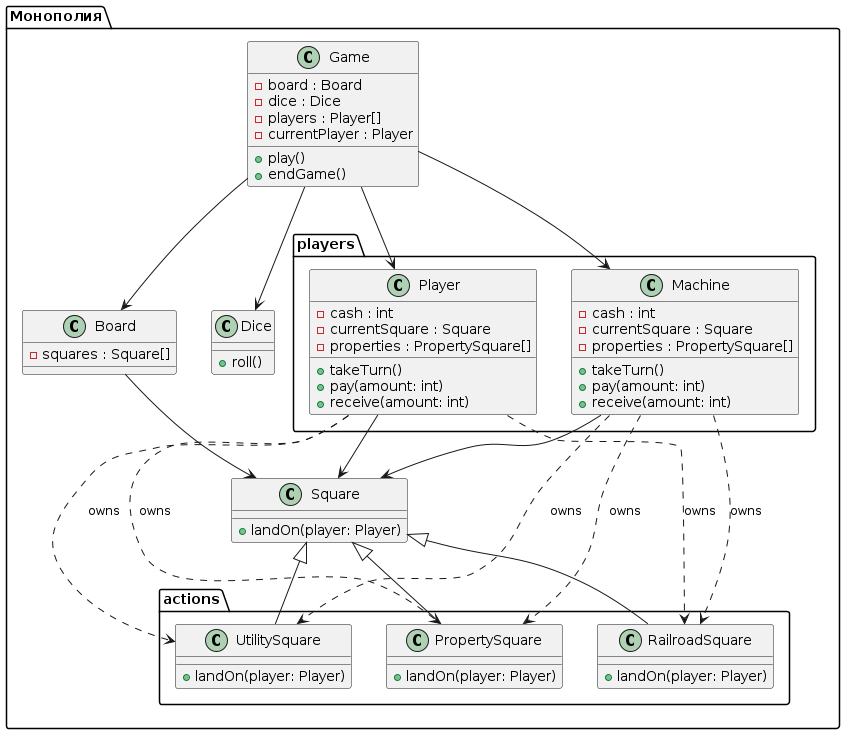


Рисунок 2 – диаграмма классов

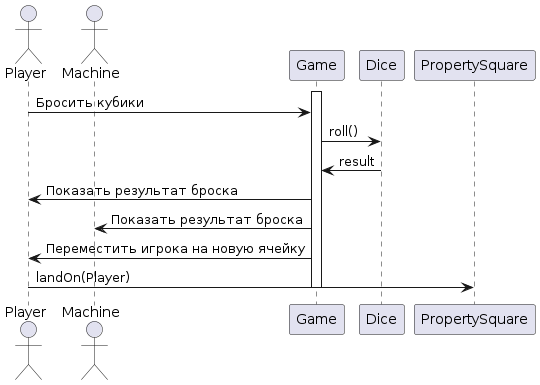


Рисунок 3 – диаграмма последовательности

Код на языке plantuml:

*Диаграмма вариантов использования:*

@startuml

package "Монополия" {

actor Player

actor Machine

rectangle "Игра монополия" {

usecase "Начать игру" as UC1

usecase "Бросить кубики" as UC2

usecase "Купить имущество" as UC3

usecase "Продать имущество" as UC4

usecase "Пройти на старт" as UC5

usecase "Закончить игру" as UC6

Player --> UC1

Player --> UC2

Player --> UC3

Player --> UC4

Player --> UC5

Player --> UC6

Machine --> UC1

Machine --> UC2

Machine --> UC3

Machine --> UC4

Machine --> UC5

Machine --> UC6

UC1 --> UC2

UC2 --> UC3

UC2 --> UC4

UC2 --> UC5

UC2 --> UC6

}

}

@enduml

*Диаграмма классов:*

@startuml

package "Монополия" {

class Game {

+play()

+endGame()

-board : Board

-dice : Dice

-players : Player[]

-currentPlayer : Player

}

class Board {

-squares : Square[]

}

class Square {

+landOn(player: Player)

}

package "actions"{

class PropertySquare {

+landOn(player: Player)

}

class UtilitySquare {

+landOn(player: Player)

}

class RailroadSquare {

+landOn(player: Player)

}

}

package "players"{

class Player {

+takeTurn()

+pay(amount: int)

+receive(amount: int)

-cash : int

-currentSquare : Square

-properties : PropertySquare[]

}

class Machine {

+takeTurn()

+pay(amount: int)

+receive(amount: int)

-cash : int

-currentSquare : Square

-properties : PropertySquare[]

}

}

class Dice {

+roll()

}

Game --> Board

Game --> Dice

Game --> Player

Game --> Machine

Board --> Square

Square <|-- PropertySquare

Square <|-- UtilitySquare

Square <|-- RailroadSquare

Player --> Square

Machine --> Square

Player ..> PropertySquare : owns

Player ..> UtilitySquare : owns

Player ..> RailroadSquare : owns

Machine ..> PropertySquare : owns

Machine ..> UtilitySquare : owns

Machine ..> RailroadSquare : owns

}

@enduml

*Диаграмма последовательности:*

@startuml

actor Player

actor Machine

participant Game

participant Dice

activate Game

Player -> Game: Бросить кубики

Game -> Dice: roll()

Dice -> Game: result

Game -> Player: Показать результат броска

Game -> Machine: Показать результат броска

Game -> Player: Переместить игрока на новую ячейку

Player -> PropertySquare: landOn(Player)

deactivate Game

@enduml