2. Class Diagram

demo.src.main.java.com.example.demo

utils

Utils

+ generatePairs(int min, int max):ArrayList<int[]>
+ shuffleIntArray(ArrayList<T> array, long seed):void

<<interface>>
com.almasb.fxgl.entity.EntityFactory

+ <methodName> (SpawnData data):Entity

<<interface>>
InitModel

+ createModel(SpawnData data):void

EntityFactory

<>
SpawnFactory

spawnData: SpawnData: SpawnData

+ spawn():void

DragonTokenFactory

- DRAGON_TOKEN_IMAGE_PATHS: String[]
- playerTurnManager: PlayerTurnManager

+ dragonTokens(SpawnData data):Entity
- createPlayerIDLabel(double x, double y, int id):Text
- createTokenImage(double x, double y, int idx)

ChitCardFactory

- coveredChitCardShapes: Circle[]
- chitCardAdapter: ChitCardAdapter

+ newChitCard(SpawnData data):Entity
- getRandomCoordinates(): List<int[]>
- createCoveredChitCardFilledShape(): Circle
- createUncoveredChirCardShape():Circle

TextFactory

+ winMsg(SpawnData data):Entity
+ turnUpdateMsg(SpawnData data):Entity
+ movementCountMsg(SpawnData data_:Entity
+ playerPointsMsg(SpawnData data): Entity

CaveFactory

- circle_radius: double

+ cave(SpawnData data):Entity
createCaveShapes(double x, double y, int i):Circle

ButtonFactory

- exitBtn: Circle
- textDesc :Text
- EXIT: ImagePattern
- EXIT_WHITE: ImagePattern

+ exitBtn(SpawnData data):Entity
+ setListeners():void

VolcanoSegment

- segmentCards: Volcano[]
- counter: int

+ addVolcanoCard(VolcanoCard volcanoCard):void
+ getSegmentCards():VolcanoCard[]

VolcanoRingFactory

- volcanoRing: Volcano[]
- segmentLength: int
- numCards: int
- numSegments: int
- rotationAmount: double

+ getVolcanoCardByID(int id):VolcanoCard
+ volcanoRing(SpawnData data):Entity
- getSegments():VolcanoSegment[]
- createVolcanoCardShape(double x, double y, VolcanoSegment segment, int j, int cardCounter)" Rectangle

ChitCardAdapter

- viewModelMapping: HashMap<Circle, ChitCard>

+ addMapping(Circle chitCardView, ChitCard chitCard): void
+ getViewControllerMapping():HashMap<Circle, ChitCard>

Controller

DragonTokenManager

- closestTokenDiff: int
- dragonTokebs: DragonToken[]
- counter: int

+ addDragonToken(DragonToken dragonToken)
+ findClosestDragonToken(DragonToken dragonToken):DragonToken
+ getClosestDragonTokenDistance():int

PlayerTurnManager

- RAND_ANIMAL_CHOICES: ArrayList<Integer>
- players: Player[]
- currPlayer: Player
- playerTurn: int
- winner: Player

+ getPlayers(): Player[]
+ getCurrPlayer(): Player
+ handleTurnTransition(): void
+ checkWinCondition: boolean
+ getWinner(): Player

TextDisplayManager

- currentTurnTextEntity: Entity
- currentMovementCount: Entity
- gameEnd: boolean

+ getInstance():TextDisplayManager
+ displayTurnUpdateMsg(int currPlayerID: Entity
+ displayWinMsg(int currPlayerID): void
+ displayMovementCountMsg(int currPlayerID, int totalMovement): Entity
+ initialize(int currPlayerID):void
+ handleEndGame(int currPlayerID):void
+ handleTurnTransition(int currPlayerID):void
+ handleMovementCountUpdate(int currPlayerID, int totalMovement)
+ displayPointsMsg(int currPlayerID, int points): void

ChitCardFlipManager

- chitCardFlipManager: ChitCardFlipManager
- coveredChitCardShapess: Circle[]
- uncoveredChitCards: ArrayList<Circle>

+ setCoveredChitCardShapes(Circle[] coveredChitCardShapes):void
+ getInstance(): ChitCardFlipManager
+ handleUncover(Circle chitCardChosen, ChitCard cardChosen):void
+ resetOnClickListener(HashMap<Circle,ChitCard> mapping):void
+ handleTurnEnd(): void

FieryDragonsApplication

- endGame: boolean
- playerTurnManager: PlayerTurnManager

initSettings(GameSettings gameSettings):void
initUI(): void
initGame():void
onUpdate():void
- handleCircleClick(Player player, Circle chitCardChosen): void
- playTurn(Player newPlayer):void

Animals

<<enum>>
AnimalType

SALAMANDER,
BABY_DRAGON,
BAT,
SPIDER,
DRAGON_PIRATE,
LEPRECHAUN

<>
Animal

- baseImgPath: String
- count: int
- animalType: AnimalType

+ getAnimal(AnimalType animalType, int count):Animal
+ fillUIShape(Shape uiShape):void
+ calculateDestinationID(Player player, DragonToken dragonToken)
- getSuffixedImgPath():String

BabyDragon

Salamander

Spider

Bat

DragonPirate

Leprechaun

+ calculateDestinationID(Player player, DragonToken dragonToken)

Model

Player

- dragonToken: DragonToken
- doNothingContinueTurn: boolean
- ID: int
- turnEnded: boolean
- animalType: AnimalType
- points: int

+ validateMove(Circle cardChosen, ChirCardAdapter chitCardAdapter)
+ moveToken(int animalCount,int destinationID, boolean isSwap): void

VolcanoCard

- ringID: int
- animalType: AnimalType
- occupied: boolean

+ setOccupied(boolean occupied):void
+ getOccupiedStatus():boolean
+ getAnimal():AnimalType
+ getRingID():int
+ setAnimal(AnimalType animal): void

DragonToken

- totalMovemenrCount: int
- initialVolcanoCardID: int
- tokenImage: Rectangle
- currentVolcanoCard: VolcanoCard
- movedOutOfCave: boolean
- playerLabel: Text
- player: Player
- currPosAngle: double

+ moveToken(int animalCount):void
+ updateTotalMovementCount(int steps):void
+ getCurrentPositionInRing(): int
+ setCurrentPosition(VolcanoCard newVolcanoCard)
+ getPlayer(): Player

ChitCard

- index: int
- animal: Animal
- coveredShape: Circle
- uncoveredShape: Circle

+ getAnimal(): Animal