MVC Architecture "Controller" GameController - gm: GameMaster **OBSERVER DESIGN OBSERVER** - displays: std::vector<SorceryDisplay*> **PATTERN** SorceryDisplay notifyDisplays(): void - notifyDisplays(MinionPtr): void notifyDisplays(int): void - notifyDisplays(string, int): void -2—— + displayMsg(): void - notifyDisplays(std::vector<string>, int): displaySorceryBoard(): void + displayHand(): void - notifyDisplays(string, int): void + displayMinion(): void + displayErrMsg(): void + go(): void // to start the game **INVOKER SUBJECT SUBJECT Sorcery UML Diagram** GameMaster Adeline Su, Julia Zhu, Sherry Feng "View" observersDisplay: std::vector<Display*> TextDisplay GraphicsDisplay - ol: ObserversList - p1: Player - gm : GameMaster* - gm: GameMaster* - p2: Player - w: Xwindow* - d1: Deck - cardwidth: Integer printPlayerBoardRow(int): void - d2: Deck - cardheight: Integer + displayMsg(string, int): void - turn: Integer // other coordinates, measurement fields + displaySorceryBoard(): void - numPlayers: int + displayHand(int): void - activePlayer: Player* + displayMinion(MinionPtr): void - nonactivePlayer: Player* + displayMsg(string, int): void + displaySorceryBoard(): void + displayHand(int): void + initPlayers(ifstream, ifstream): void + displayMinion(MinionPtr): void + startTurn(): void + endTurn(): void + attackMinion(int, int): void + attackPlayer(int): void + activateAbility(): void + discard(): void + useAbility(int): void + play(int): void // update observers if needed XWindow + notifyStartTurnObservers(): void + notifyEndTurnObservers(): void - d: Display* + notifyMinionEnterObservers(): void - w: Window + notifyMinionLeaveOObservers(): void - gc: GC + attatch(TriggeredAbility*): void + detach(TriggeredAbility*): void + fillRectangle(int, int , int, int, int): void + drawString(int, int, string) Card - CardType: enum class //Spell, Minion, Enchantment or Ritual - CardName: enam class // Name of all cards Player - name: CardName - cost: Integer name: String - desc: String - id: Integer - needTarget: Boolean · life: Integer - magic: Integer - deck: Deck // various getters hand: Hand ObserversList - board: Board - grave: Graveyard + activePlayer: Player* ritual: RitualPtr + observers std::vector<TriggeredAbility> + init(string, int, ifstream, ObserversList): void **DECORATOR** + onBoard(MinionPtr): Boolean **DESIGN PATTERN** + increaseMagic(int): void Spell Minion Enchantment Ritual + increaseLife(int): void + deceaseLife(int): void - activationCost: Integer - aa: ActivatedAbility* type: CardType type: CardType + drawCard(): CardPtr - needTarget: Boolean - needTarget: Boolean - charge: Integer + destroyRitual(): void - board: Board* attackModifer: String trigAbility: TriggeredAbility* + applyAbility(Player&, Player&): void + removeTrigger(TriggeredAbility*): void - desc: String defenseModifier: String + applyAbility(Player&, Player&, int): void + play(int, Player&): TriggeredAbility + decreaseCharge(int): void + play(int, int, Player&): void + destroy(): void + increaseCharge(int): void + useAbility(int, Player&): void + isDead(): Boolean + trigger(): void + useAbility(int, int, Player): void // + various accessor methods ActivatedAbility **SUBJECT** - activationCost: Integer Deck Hand **Board** Graveyard - hitBoth: Boolean - needTarget: Boolean DefaultMinion - theDeck, std::vector<CardPtr> - theHand, std::vector<CardPtr> - theBoard, std::vector<MinionPtr> - theGrave: std::Stack<Minion*> - grave: grave* - cardName: CardName - doEffect(Player&, int): void ol: ObserversList* + init(ifstream, Player): void + init(Deck&): void + isEmpty(): Boolean - desc: String + applyAbility(Player&, Player&, int): void + getTop(): MinionPtr + shuffle(): void + addCard(CardPtr): void - attack: Integer + incActivationCost(int): void + drawCard(), CardPtr + init(ObserversList*): void + removeTop(): void + restoreAction(): void defense: Integer + removeCard(int): void + addCard(MinionPtr): void + push(MinionPtr): void action: Integer EnchantmentDec + removeCard(int): void ability: variant<ActivatedAbility*, + enchantMinion(int, string, int): void TriggeredAbility*, monostate> - hidden: Boolean + stripEnchants(int): void - ta: TriggeredAbility* + stripTopEnchant(int): void # next, MinionPtr // + various accessor methods BanishAbility UnsummonAbility RechargeAbility DisenchantAbility RaiseDeadAbility + restoreAction(): void + attach(TriggeredAbility*): void - doEffect(Player&): void - doEffect(Player&): void doEffect(Player&): void doEffect(Player&): void - doEffect(Player&): void + detach(TriggeredAbility*): void // + various accessor methods + notifyMinionEnterObservers(MinionPtr): void + notifyMinionLeaveObservers(MinionPtr): void BlizzardAbility ApprenticeSummonerAbility MasterSummonerAbility - doEffect(Player&): void - doEffect(Player&): void doEffect(Player&): void doEffect(Player&): void **COMMAND DESIGN** PATTERN COMMAND **OBSERVER DESIGN** GiantStrength Enrage Haste MagicFatigue Silence **OBSERVER** PATTERN TriggeredAbility // + accessors | // + accessors | // + accessors | // + accessors - TriggerType: enum class - TriggerCardType: enum class type: TriggerType cardType: TriggerCardType - owner: Player* - ownerMinion: MinionPtr - targetPlayers, std::vector<Player*> - targetMinions, std::vector<MinionPtr> // receivers + applyAbility(), void // notify, execute // + various mutator methods

AuraofPowerAbility

+ applyAbility(): void

StandstillAbility

+ applyAbility(): void

BoneGolemAbility

+ applyAbility(): void

FireElementalAbility

+ applyAbility(): void

+ applyAbility(): void

DarkRitualAbility

+ applyAbility(): void

PotionSellerAbility

+ applyAbility(): void

main()

- controller: GameController