Implementing a Hearthstone Engine

Using Model-Driven Development Approach

Bart Cools

6 September 2018

Outline

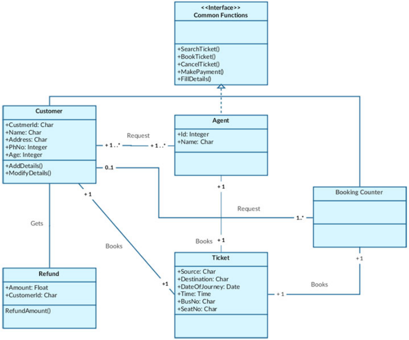
1. Background
   - Concepts
   - Tools
2. Related Works
3. Experimental Setup
   - Scope
   - Model
   - Transformation
4. Results
5. Conclusion

Outline

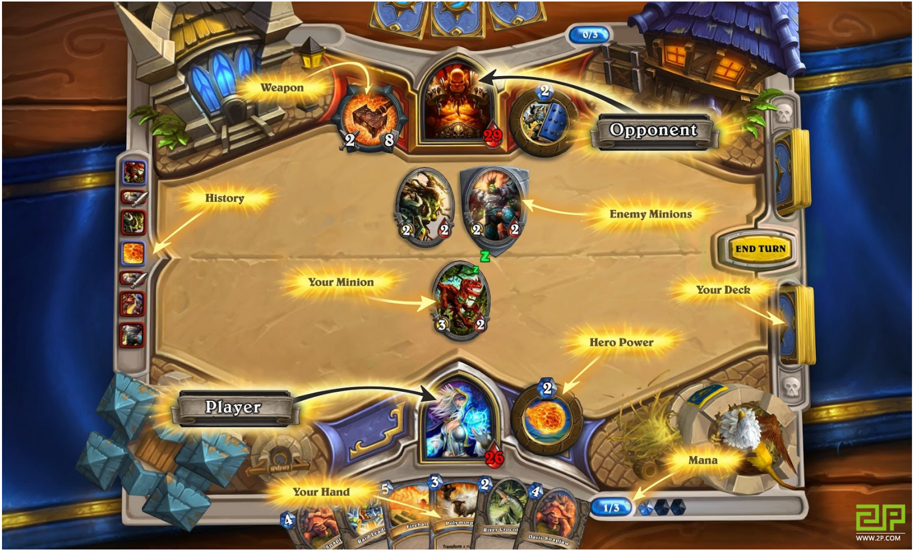
1. Background
   - Concepts
   - Tools
2. Related Works
3. Experimental Setup
   - Scope
   - Model
   - Transformation
4. Results
5. Conclusion

Model-Driven Software Design

- Higher level of abstraction
- Specify data, attributes and relations instead of writing code



Trading Card Games

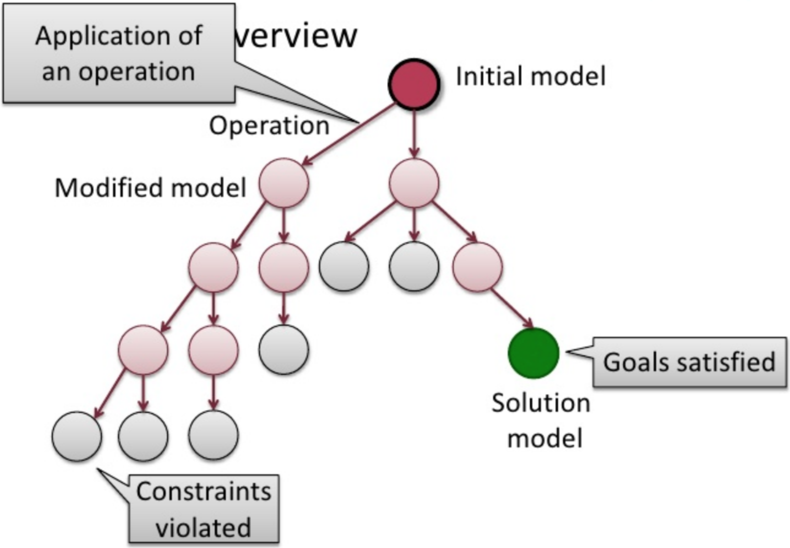


Game Engines

VIATRA

- Eclipse Framework
- VIsual Automated model TRAnsformations
- Query engine and Model Transformations
- Design Space Exploration for game engine

Design Space Exploration



Outline

1. Background
   - Concepts
   - Tools
2. Related Works
3. Experimental Setup
   - Scope
   - Model
   - Transformation
4. Results
5. Conclusion

Current state of Research

- Few attempts found for model-driven game development
- Object-oriented game engine found for popular board games
- Simulators for Hearthstone exist
- No model-driven game engine found

Outline

1 Background
- Concepts
- Tools

2 Related Works

3 Experimental Setup
- Scope
- Model
- Transformation

4 Results

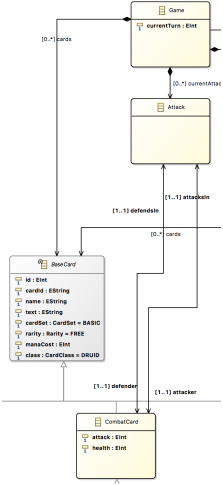
5 Conclusion

Scope

- Building all mechanics of Hearthstone not doable in short period
- Implement small subset of the game
- Proof by construct for feasibility
- All mechanics adhere to the same Phase/Queue rules
- We modeled the Attack Into Death Resolution mechanic

Model

- Model attack mechanic at a higher level
- Create class diagram
- Multiple changes throughout process
- Generates robust production-level Java code



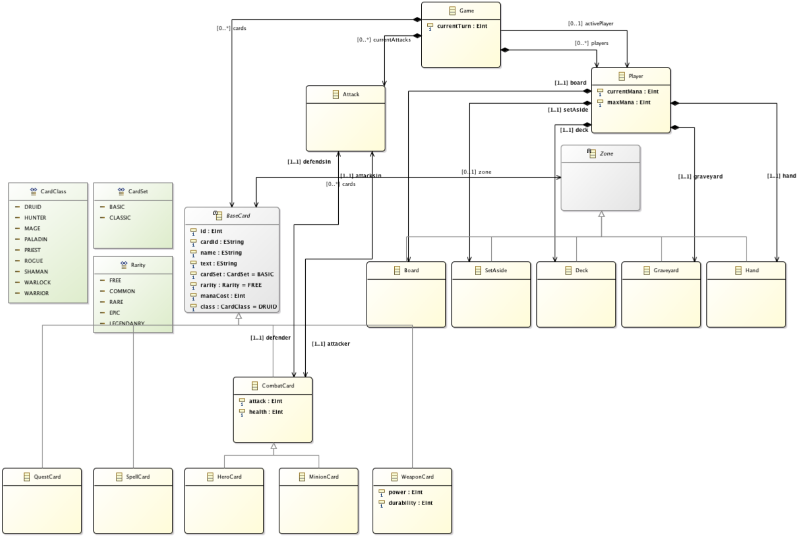
Query and Transformation

- VIATRA Query Language to define queries
- Queries designed in declarative manner
- Xtend for transformation rules
- Query and Transformations tightly coupled

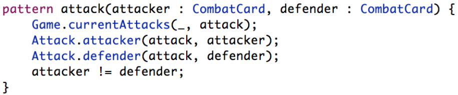
Outline

1. Background
   - Concepts
   - Tools
2. Related Works
3. Experimental Setup
   - Scope
   - Model
   - Transformation
4. Results
5. Conclusion

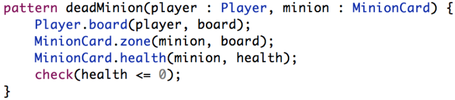
Model


Game
currentTurn : EInt
[0..*] cards
[0..*] currentAttacks
[0..1] activePlayer
[0..*] players
Player
currentMana : EInt
maxMana : EInt
[1..1] board
[1..1] setAside
[1..1] deck
Attack
[1..1] defendsIn
[1..1] attacksIn
[0..*] cards
[0..1] zone
Zone
[1..1] graveyard
[1..1] hand
CardClass
DRUID
HUNTER
MAGE
PALADIN
PRIEST
ROGUE
SHAMAN
WARLOCK
WARRIOR
CardSet
BASIC
CLASSIC
Rarity
FREE
COMMON
RARE
EPIC
LEGENDARY
BaseCard
id : EInt
cardId : EString
name : EString
text : EString
cardSet : CardSet = BASIC
rarity : Rarity = FREE
manaCost : EInt
class : CardClass = DRUID
Board
SetAside
Deck
Graveyard
Hand
[1..1] defender
[1..1] attacker
CombatCard
attack : EInt
health : EInt
QuestCard
SpellCard
HeroCard
MinionCard
WeaponCard
power : EInt
durability : EInt

Queries



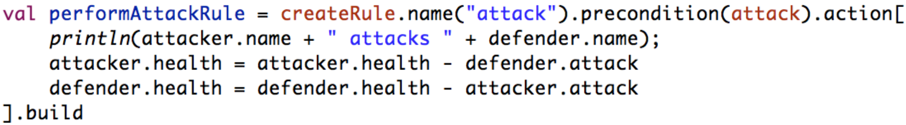
```
pattern attack(attacker : CombatCard, defender : CombatCard) {
    Game.currentAttacks(_, attack);
    Attack.attacker(attack, attacker);
    Attack.defender(attack, defender);
    attacker != defender;
}
```



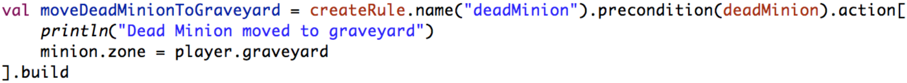
```
pattern deadMinion(player : Player, minion : MinionCard) {
    Player.board(player, board);
    MinionCard.zone(minion, board);
    MinionCard.health(minion, health);
    check(health <= 0);
}
```

- Uses pattern matching to find objects
- Easily transformed from Java code

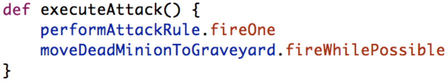
Transformation Rules



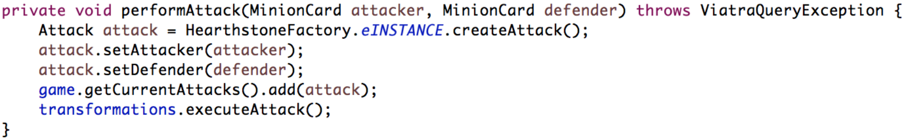
```
val performAttackRule = createRule.name("attack").precondition(attack).action[
    println(attacker.name + " attacks " + defender.name);
    attacker.health = attacker.health - defender.attack
    defender.health = defender.health - attacker.attack
].build
```



```
val moveDeadMinionToGraveyard = createRule.name("deadMinion").precondition(deadMinion).action[
    println("Dead Minion moved to graveyard")
    minion.zone = player.graveyard
].build
```

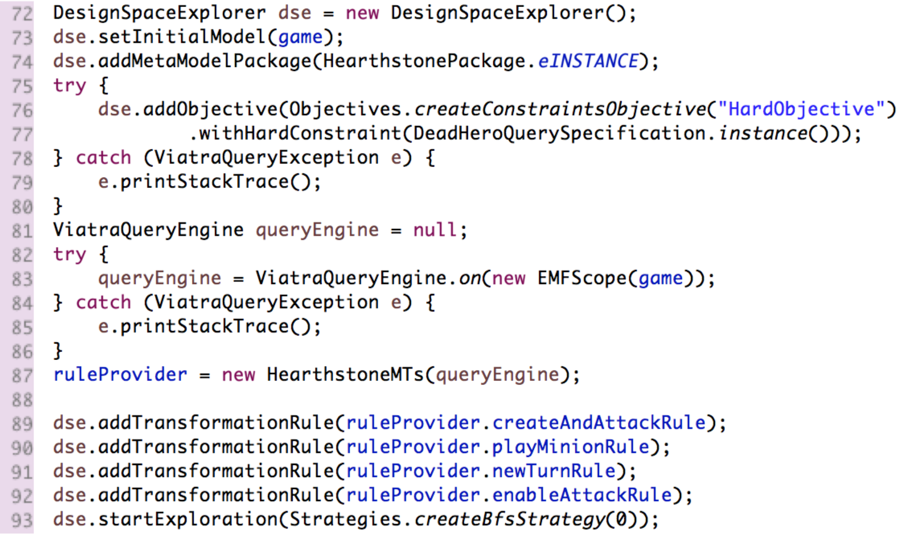


```
def executeAttack() {
    performAttackRule.fireOne
    moveDeadMinionToGraveyard.fireWhilePossible
}
```



```
private void performAttack(MinionCard attacker, MinionCard defender) throws ViatraQueryException {
    Attack attack = HearthstoneFactory.eINSTANCE.createAttack();
    attack.setAttacker(attacker);
    attack.setDefender(defender);
    game.getCurrentAttacks().add(attack);
    transformations.executeAttack();
}
```

- BatchTransformation vs EventDrivenTransformation

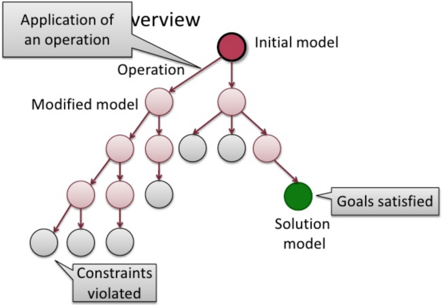


```
DesignSpaceExplorer dse = new DesignSpaceExplorer();
dse.setInitialModel(game);
dse.addMetaModelPackage(HearthstonePackage.eINSTANCE);
try {
    dse.addObjective(Objectives.createConstraintsObjective("HardObjective")
            .withHardConstraint(DeadHeroQuerySpecification.instance()));
} catch (ViatraQueryException e) {
    e.printStackTrace();
}
ViatraQueryEngine queryEngine = null;
try {
    queryEngine = ViatraQueryEngine.on(new EMFScope(game));
} catch (ViatraQueryException e) {
    e.printStackTrace();
}
ruleProvider = new HearthstoneMTs(queryEngine);

dse.addTransformationRule(ruleProvider.createAndAttackRule);
dse.addTransformationRule(ruleProvider.playMinionRule);
dse.addTransformationRule(ruleProvider.newTurnRule);
dse.addTransformationRule(ruleProvider.enableAttackRule);
dse.startExploration(Strategies.createBfsStrategy(0));
```

Design Space Exploration

- Initial Model: The State of the game you want to analyse
- Operation: The Transformation Rules applied to the State
- Solution Model: Model that satisfies all constraints



Outline

1. Background
   - Concepts
   - Tools
2. Related Works
3. Experimental Setup
   - Scope
   - Model
   - Transformation
4. Results
5. Conclusion

Conclusion

- It is feasible to model an AI trading card games
- Iterative process of developing new mechanics
- Scope → model → query → transformation
- Bundle into DSE to create prototype of an Engine

What's next

- Extend the modeled game to all the mechanics
- Model History for better heuristics and pruning
- Research Internship 2: Literature study on the artificial intelligence for games. Focus on elements of chess, Go and Texas Hold'em
- Master Thesis: Combine model-driven approach and literature study to build Hearthstone engine and benchmark the results