Projet de CPS : Lemmings

Morvan Lassauzay et Victor Nea

Les différents services

- Level : Spécification + contrat + tests
- GameEng : Spécification
- Lemming : Spécification
- Player : Spécification

Service Level

Types: bool, int, enum Nature{EMPTY, DIRT, METAL, ENTRANCE, EXIT}

Observators:

```
const height: [Level] -> int
const width: [Level] -> int
editing: [Level] -> bool
nature: [Level] * int * int -> Nature
obstacle: [Level] * int * int -> bool
squareExist: [Level] * int * int -> bool
hEntrance: [Level] -> int
wEntrance: [Level] -> int
```

Constructors:

```
init: int * int -> [Level]
```

Operators:

```
setNature: [Level] * int * int * Nature -> [Level]
remove: [Level] * int * int -> [Level]
build: [Level] * int * int -> [Level]
goPlay: [Level] * int * int *int *int -> [Level]
```

Service GameEng

Types: bool, int

Observators:

score: [GameEng] -> int
turn: [GameEng] -> int
const sizeColony: [GameEng] -> int
nbLemmingsDead: [GameEng] -> int
nbLemmingsSaved: [GameEng] -> int
nbLemmingsActive: [GameEng] -> int
nbLemmingsCreated: [GameEng] -> int
lemming: [GameEng] * int -> Lemming
active: [GameEng] * int -> bool
numLemmingsActive: [GameEng] -> Set<int>
const spawnSpeed: [GameEng] -> int
gameOver:[GameEng] -> bool
level: [GameEng] -> Level
Const levelInit : [GameEng] -> Level

Constructors:

init: int * int * Level -> [GameEng]

Operators:

executeTurn: [GameEng] -> [GameEng]

Service Lemming

Types: bool, int,
enum Direction{LEFT, RIGHT},
enum Behaviour{WALKER, FALLER, DIGGER, BUILDER, STOPPER, BASHER},
enum State{BASIC, CLIMBER, FLOATER, BOMBER}

Observators:

const num: [Lemming] -> int
direction: [Lemming] -> Direction
behaviour: [Lemming] -> Behaviour
state: [Lemming] -> State
hPos: [Lemming] -> int
wPos: [Lemming] -> int
counterFaller: [Lemming] -> int
dead: [Lemming] -> bool
level: [Lemming] -> Level

Constructors:

init: int * int * int * Level -> [Lemming]

Operators:

setBehaviour: [Lemming] * Behaviour -> [Lemming]
setState: [Lemming] * State -> [lemming]
step: [Lemming] -> [Lemming]

Service Player

Types: bool, int,enum TokenType{WALKER, DIGGER, BUILDER, STOPPER, BASHER, BASIC, CLIMBER, FLOATER, BOMBER}

Observators:

nbTokenInit: [Player] * TokenType -> int
nbToken: [Player] * TokenType -> int
tokenSelected: [Player] -> TokenType
gameEngine: [Player] -> GameEng

Constructors:

init: int * int * int * int * int * int * int * int * int * int * GameEng -> [Player]

Operators:

useToken: [Player] * int -> [Player]
resetGame: [Player] -> [Player]
selectToken: [Player] * TokenType -> [Player]

Organisation du projet

- buggycomponents
- components
- contracts
  - ContractError.java
  - GameEngContract.java
  - InvariantError.java
  - LemmingContract.java
  - LevelContract.java
  - PlayerContract.java
  - PostconditionError.java
  - PreconditionError.java
- decorators
- enums

- main
- services
  - display
  - GameEngService.java
  - LemmingService.java
  - LevelService.java
  - PlayerService.java
  - RequireGameEngService.java
  - RequireLemmingService.java
  - RequireLevelService.java
  - RequirePlayerService.java
- > tests

Test-MBT

Critères de couverture appliqués :

- Préconditions (levée d'une PreconditionError)
- Transitions (PostconditionError ou InvariantError)
- États intéressants (Confirmation d'un état attendu)
- Scénarios utilisateurs (Confirmation d'un état attendu)

Exemple de test : initPre

- Test : testInitPre1

Coverage : preconditions
Goal : precondition resolved
Initial conditions : empty
Operations : C0 := init(MAX_HEIGHT, MAX_WIDTH)
Oracle :
    No exception
Report :
    A PreconditionError was raised :
        "No exception must be raised."

- Test : testInitPre2

Coverage : preconditions
Goal : precondition unresolved
Initial conditions : empty
Operations : C0 := init(MIN_HEIGHT-1, 30)
Oracle :
    A PreconditionError must be raised
Report :
    No exception :
        "A PreconditionError must be raised."

Conclusion

- Expressivité du langage de spécification
- Méthodologie efficace
- Mise à jour compliquée