Meta Game Language

Author:
Sune Chung JEPSEN

Supervisor:
Ulrik Pagh SCHULTZ

May 2018

SDU

Contents

1 The domain and purpose of the DSL

The domain is to create a DSL where users can define a 2D top down game. Through simple commands for defining "Objects", "Locations", "Actions" and "Executions" a game can be stated and executed through a simple graphical interface as seen in figure 1.

Figure 1: Example of a 2D top down game

The user can write a limited set of instructions, without heavy programming experience and get a game up and running. All the functionality is based on a DSL and an extensive manual coded gameframework as described in appendix section B.6.

2 Metamodel

Figure 2 is an overview of the most important elements in the metamodel. The metamodel will be explained by example as seen in figure 3. The game is a Labyrinth, where an agent should navigate to a goal with least possible steps. Towards the goal there are walls which the agent should navigate around.
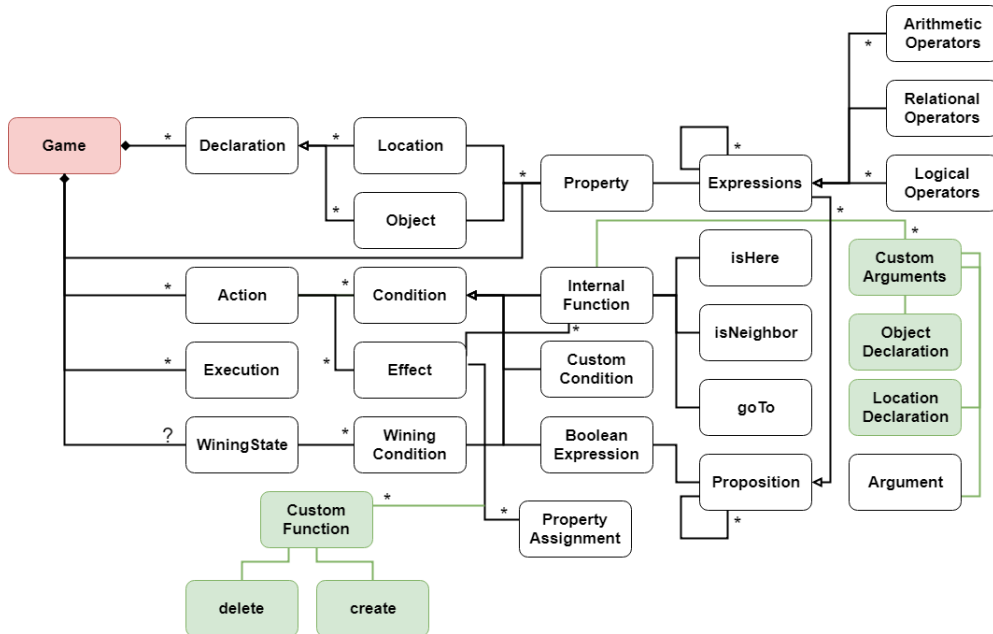

Figure 2: The meta model, where the green figures are the extensions.

First a "Game" is defined. A "Game" consist of "Declarations" which are shown in figure 3 (1). "Declaration" defines the "Objects" and the "Locations" which are placed on the game board through coodinates. Both "Objects" and "Locations" might have "Properties". A "Game" can also contain "Properties" (Fields). The grammar for these rules can be seen in appendix A.1.1 listing 10 line 1-2.

A "Game" consist of "Actions" and a "WinningState" which is shown in figure 3 (2). A "Move" "Action" is defined which consist of a "Condition" and an "Effect". The condition states the player should be an agent, the next location cant be a wall and the next location is neighbor to the current players position (InternalFunction). If the condition is fulfilled an effect will be executed, which is the actual move and the players path property will be incremented. The grammar for these rules can be seen in appendix A.1.1 listing 10 line 33-35 ("Action").

Furthermore a "WinningState" together with "WiningCondition" is defined. To win requires that the "Agent" is placed on the "Goal" and that the steps taken by the "Agent" are less then a predefined variable (Field). The grammar for these rules can be seen in appendix A.1.1 listing 10 line 132-134 and 136-138 ("WinningState" and "WinningConditions").

Note that a both "Game", "Objects" and "Location" can have "Properties" which might contains "Expressions". An "Expression" might be an "Variable" or a "LocaleVariable". Game defines "Variables" and "Object" / "Location" defines "LocaleVariables". The grammar for these rules can be seen in appendix A.1.1 listing 10 line 172-174 ("LocaleVariable" and "Variable").

Last the "Executions" are defined as seen in figure 3 (3). The "Execution" is where the game is played. In this example the "Agent" is calling the "Move" "Action" four times. The grammar for "Execution" can be seen in appendix A.1.1 listing 10 line 41-43. An optimized presentation of the game can be seen in appendix C.2 figure 19.

1
```
Game LabyrintQuest

number maxPathLength = 11

Object Agent (0,0)
    truth value isAgent = true
    number path = 0
Location Wall (1,0) (2,1) (2,2) (1,2)
    truth value isWall = true
Object Goal (2,0)
```

2
```
Action Move (player,next)
    Condition player.isAgent, !next.isWall, isNeighbor(arg player,arg next)
    Effect goTo(arg player,arg next), player.path++

WinningState isHere(objdec Agent, objdec Goal), Agent.path <= maxPathLength
```

3
```
Move(Agent, (0,1))
Move(Agent, (0,2))
Move(Agent, (0,3))
Move(Agent, (1,3))
```

Figure 3: LabyrinthQuest game.

The "green" extensions of the metamodel will be explained in section 3.2.1. The full grammar can be seen in appendix A.1.1 listing 10.

3 Extensions

Validation functionality has been develop to prevent unpredicted behavior e.g. is should not be possible to use properties on "Objects" or "Locations" which are not defined.

The grammar has been extended with delete of "Objects" and "Locations" functionality. The delete functionality makes it possible to create new creative

games. The user will be able to assign "Actions" with delete functionality as an "Effect".

3.1 Validation

Every validation task will have a screendump of the DSL program which illustrates the validation in action. To avoid losing overview the code snippets will be short and concise. Common techniques are used for all validation such as; factoring to helper methods, dispatch methods and switch expression with type guard.

For validation rules in section 3.1.1, section 3.1.2 and section 3.1.4, "Expression" are needed to get information about the "LocaleVariables" and "Variables", therefor extending the "getVariables" method was needed as seen in appendix A.1.2 listing 11.

3.1.1 Game Field Object/Location Name and Property

Following validation holds for properties on the "Game".

Left image figure 4, shows it should not be possible to use an "Object" or a "Location" which isn't defined.

Right image figure 4, shows it should not be possible to use a "Property" on an "Object" or a "Location" which do not exist.
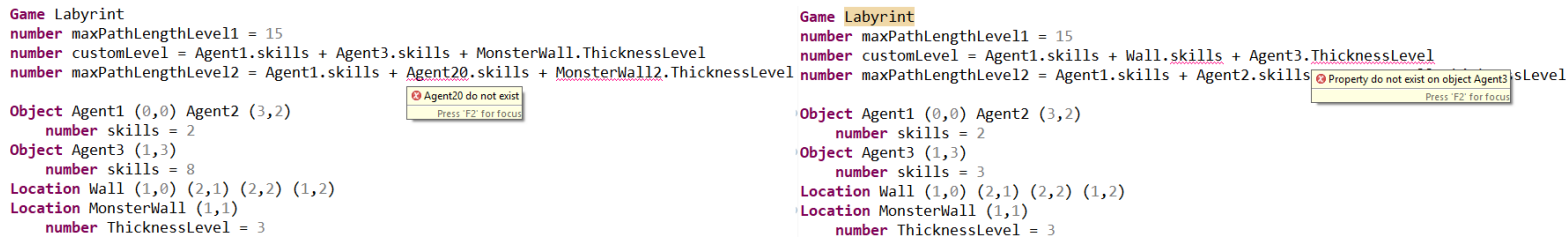

Figure 4: Validation on Game Field Object/Location Name and Property

```
1   @Check
2   def validateGameFieldObjectLocationNameProperty(Game game) {
3     for (Property property : game.fields) {
4       property.validateObjectLocationProperty(game, "validateGameFieldObjectProperty")
5       property.validateObjectLocationName(game, "validateGameFieldObjectProperty")
6     }
7   }
```

Listing 1: Validation code Game Field Object/Location Name and Property

3.1.2 Object/Location Name and Property on Declarations

Following validation holds for properties on "Objects" and "Locations".

Left image figure 5, shows it should not be possible to use an "Object" or a "Location" which isn't defined.

Right image figure 5, shows it should not be possible to use a "Property" on an "Object" or a "Location" which do not exist.

```
Object Agent1 (0,0) Agent2 (3,2)
    number skills = Agent3.skills + Agent33.skills + MonsterWall22.ThicknessLevel
Object Agent3 (1,3)                        [⊗ MonsterWall22 do not exist]
    number skills = 3                      [        Press 'F2' for focus]
Location Wall (1,0) (2,1) (2,2) (1,2)
    number ThicknessLevel =1
Location MonsterWall (1,1)
    number ThicknessLevel = 3
```

```
Object Agent1 (0,0) Agent2 (3,2)
    number skills = Agent3.skills + MonsterWall.skills + Agent3.ThicknessLevel
Object Agent3 (1,3)                        [⊗ Property do not exist on location MonsterWall]
    number skills = 3                      [              Press 'F2' for focus]
Location Wall (1,0) (2,1) (2,2) (1,2)
    number ThicknessLevel = Agent3.skills + Agent2.skills + MonsterWall.ThicknessLevel
Location MonsterWall (1,1)
    number ThicknessLevel = 3
```

Figure 5: Validation on Object/Location Name and Property on Declarations

```
1   @Check
2   def validateObjectLocationNameProperty (Game game) {
3      for (Declaration declaration : game.declarations) {
4        for (Property property : declaration.properties) {
5          property.validateObjectLocationProperty(game, "validateObjectLocationProperty")
6          property.validateObjectLocationName(game, "validateObjectLocationName")
7        }
8      }
9   }
```

Listing 2: Validation code for Object/Location Name and Property on Declarations

3.1.3 Duplicate Object Location Names and Property on Declarations

Following validation holds for "Objects" and "Locations".

Left image figure 6, shows it should not be possible to define duplicate "Object" or a "Location" names.

Right image figure 6, shows it should not be possible to define duplicate "Properties" on either "Objects" or "Locations".

```
Object Agent1 (0,0) Agent1 (3,2) Agent1 (0,2)
    number skills = 2
Object Agent1(3,3) Agent2(3,1)     [⊗ Object Agent1 is a duplicate]
    number skills = 3              [       Press 'F2' for focus]
Location MonsterWall (1,1)  MonsterWall (1,2)
    number ThicknessLevel = 3
Location MonsterWall (2,1)  BrickWall (2,3)
    number ThicknessLevel = 3
```

```
Object Agent1 (0,0) Agent2 (3,2)  Agent3 (3,2)
    number skills = 2
    number skills = 2
    number      [⊗ Property skills already exist on object]
Location Mo     [      Press 'F2' for focus] terWall2 (1,2)
    number ThicknessLevel = 3
    truth value isWall = true
    truth value isWall = true
Location MonsterWall3 (2,1)  BrickWall1 (2,3)
    truth value isWall = true
    number ThicknessLevel = 10
```

Figure 6: Validation on duplicate Object/Location Names and Property on Declarations

```
1    @Check
2    def validateDuplicateObjectLocationNameProperty (Game game) {
3       var List<String> seenObjectList = new ArrayList()
4       var List<String> seenLocationList = new ArrayList()
5       for (Declaration declaration : game.getDeclarations) {
6          declaration.validateProperty
7          switch declaration {
8            Object: {
9               for (ObjectDeclaration objectDeclaration : declaration.declarations) {
10                 if (seenObjectList.contains(objectDeclaration.name)) {
11                    error("Object " + objectDeclaration.name + " is a duplicate ",
     objectDeclaration ,
```

```
12                   MetaGameLanguagePackage . Literals .OBJECT_DECLARATION__NAME) ;
13              } else {
14                seenObjectList . add ( objectDeclaration . name)
15              }
16            }
17          }
18        Location : {
19          for ( LocationDeclaration locationDeclaration : declaration . declarations ) {
20            if ( seenLocationList . contains ( locationDeclaration . name ) ) {
21              error ( "Location " + locationDeclaration . name + " is a duplicate ",
      locationDeclaration ,
22                   MetaGameLanguagePackage . Literals .LOCATION_DECLARATION__NAME) ;
23            } else {
24              seenLocationList . add ( locationDeclaration . name)
25            }
26          }
27        }
28      }
29    }
30  }
```

Listing 3: Validation code for duplicate Object/Location Names and Property on Declarations

3.1.4 Validate Circular References

Figure 7 shows it should not be possible to define circular references on "Object-", "Location-" or "Game" "Properties".

Figure 7: Validation Circular References on Objects-, Locations- and Fields Properties

First a datastructure is build which contains all variables and their dependencies. Then the datastructure is traversed as follows:

1. Check if child is a "LocalVariable" or a "Variable"

2. If "LocalVariable" -> Get all "Expression" for current child

   (a) Loop through all "Expression" for current child

   (b) Check if each of the "Expressions" is a "LocalVariable" or a "Variable"

   (c) If "LocaleVariable" -> If seen "before" then throw error

       i. If seen list doesn't contain "LocaleVariable" -> call itself (child=expression, parent=child)

   3. If "Variable" -> Get all "Expression" for current child

   4. ... Same procedure for "Variable" ...

The core functionality contains two methods for traversing the dependencies as shown in listing 4.

```
1    private var Map<String, List<Expression>> graph = new HashMap()
2    private var List<String> seen = new ArrayList();
3    @Check
4    def validateCircularReferences(Game game) {
5        // Build dependency graph between variables
6        ...
7        validate(child, entry.key, seen)
8    }
9    def boolean validate(Expression child, String parent, List<String> seen) {
10       // Do recursive call
11   }
```

Listing 4: Methods for validating Circular Reference

It should be noted that concepts behind the design is discussed with group member Anna Ølgaard Nielsen.

3.2 Grammar extension

3.2.1 Delete Objects/Location functionality

To achieve delete functionality, core functionality of the "Effects" had to be extended as follows:

1. A new "CustomFunction" has been added to the "CustomEffect" rule.

2. A new rule, "CustomFunction", with "Arguments"

3. A new rule, "CustomFunctionName", which contains keywords "delete" and "create".

4. "InternalFunction" takes "CustomArgements" as arguments. (Before it only took "Arguments")

All extended rules can be seen in listing 5. Two dispatch methods; Xtend code for "InternalFunction" and "CustomArguments" are shown in listing 6 and Xtend code for deleting "Object" or "Location" is shown in listing 7. Two listing of generating code; delete of "Object" or "Location" is shown in listing 8 and an internal function in listing 9.

The internal functions are extended to take parameters "Argument", "ObjectDeclaration" or a "LocationDeclaration", as references. The purpose for this extension is to give the opportunity to state a delete "Action" of "Objects" or "Locations" together with a "Condition". The "Condition" might be that the delete "Action" do not have the ability to delete a certain "Goal" (Object/Location) condition such as "!isHere(objdec Goal, arg next)" as shown in listing 9.

6

With the existing inference functionality, inference is based on "Properties" from an existing "Object" or "Location" "Property" in the "Declaration" structure. Since the arguments with their corresponding properties do not need to be used elsewhere, there was a need to introduce this extension.

```
1  Effects :
2    effects+=CustomEffects (',' effects+=CustomEffects)*
3  ;
4
5  CustomEffects :
6    PropertyAssignment | InternalFunction | CustomFunction
7  ;
8  CustomFunctionName :
9    'delete' | 'create'
10 ;
11
12 CustomFunction :
13   internal_name=CustomFunctionName '(' arguments=Arguments? ')'
14 ;
15
16 InternalFunction :
17   not='!'? internal_name=InternalName '(' arguments=CustomArguments? ')'
18 ;
19
20 CustomArguments :
21   arguments+=ArgumentOrObjectOrLocation (',' arguments+=ArgumentOrObjectOrLocation)*
22 ;
23
24 ArgumentOrObjectOrLocation :
25   {Arg}'arg' arg=[Argument]| {Objdec}'objdec' objdec=[ObjectDeclaration]| {Locdec}'locdec
         ' locdec=[LocationDeclaration]
26 ;
```

Listing 5: Grammar for Delete Objects/Location functionality

```
1
2  def dispatch CharSequence generateEffect(InternalFunction f){
3      var names = new ArrayList<String>()
4      for (arg:f.arguments?.arguments) {
5        switch arg {
6          Arg: names.add(arg.arg.name)
7          Objdec: names.add("model.getObject(\""+arg.objdec.name+"\")")
8          Locdec: names.add("model.getLocation(\""+arg.locdec.name+"\")")
9        }
10     }
11   ... Generate code for internal function ...
12   }
```

Listing 6: Minified Xtend code for Internal function. Same code is used for generateWinCondition and generateCondition since both might be an InternalFunction

```
1   def dispatch CharSequence generateEffect(CustomFunction cf) '''
2      // delete object in metamodel
3
4      // delete location in metamodel
5      '''
```

Listing 7: Minified Xtend code for delete

```
1
2  // delete in object
3  List<Object>objectsToRemove = new ArrayList<>();
4  for(int i=0; i<model.getAllObjects().size(); i++){
5      if(model.getAllObjects().get(i).getPosition().X == next.getPositions().get(0).X &&
6              model.getAllObjects().get(i).getPosition().Y == next.getPositions().get(0).Y
      &&
7              model.getAllObjects().get(i).getName() != player.getName()){
8          objectsToRemove.add(model.getAllObjects().get(i));
9      }
10 }
11 model.getAllObjects().removeAll(objectsToRemove);
12
13 // delete in location
14 List<Position>positionsToRemove = new ArrayList<>();
15 for(int i=0; i<model.getAllLocations().size(); i++){
16     for(int y=0; y<model.getAllLocations().get(i).getPositions().size(); y++){
17         if(model.getAllLocations().get(i).getPositions().get(y).X == next.getPositions().
      get(0).X &&
18                  model.getAllLocations().get(i).getPositions().get(y).Y == next.
      getPositions().get(0).Y ){
19             positionsToRemove.add(model.getAllLocations().get(i).getPositions().get(y));
20         }
21     }
22     model.getAllLocations().get(i).getPositions().removeAll(positionsToRemove);
23 }
```

Listing 8: Generated code for delete

```
1  public boolean execute(Map<String, java.lang.Object> args, MachineMetaModel model) {
2  ....
3      executionAllowed &= !_iInternalFunction.isHere(model.getObject("Goal"), next);''
4  ....
5  }
```

Listing 9: Minified generated code for Internal function

References

[BCK12] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice, 3rd ed.* Addision-Wesley, 2012.

[Chr10] Henrik Bærbak Christensen. *Flexible, reliable software: Using Patterns and Agile Development.* Chapman & Hall, 2010.

[Fow10] Martin Fowler. *Domain Specific Languages.* Addison-Wesley Professional, 1st edition, 2010.

Appendices

A   Individual rapport

A.1   Listings

A.1.1   Grammar

```
Game:
  'Game' name=ID fields+=Property* declarations+=Declaration* actions+=Action*
    winningState=WinningState? executions+=Execution*
;

Declaration:
    Object | Location
;

Location:
  'Location' declarations+=LocationDeclaration+ properties+=Property*
;

LocationDeclaration:
  name=ID coordinates+=Coordinates+
;

Object:
  'Object' declarations+=ObjectDeclaration+ properties+=Property*
;

ObjectDeclaration:
  name=ID coordinates=Coordinates
;

Coordinates:
  '(' 'x=INT' ',' 'y=INT' ')'
;

Property:
  {BoolExp} 'truth' 'value' name=ID '=' bool_exp=BooleanExp | {NumberExp} 'number' name=
    ID '=' math_exp=MathExp
;

Action:
  declaration=ActionDeclaration condition=ConditionDeclaration effect=EffectDeclaration
;

ActionDeclaration:
  'Action' name=ID '(' args=Arguments? ')'
;

Execution:
  action_name=[ActionDeclaration]   '(' executionArgs=ExecutionArg? ')'
;

ExecutionArg:
  executionArgs+=ExecutionDeclaration (',' executionArgs+=ExecutionDeclaration)*
;

ExecutionDeclaration:
  Argument | Coordinates
;
```

```
52
53 Arguments:
54   arguments+=Argument (',' arguments+=Argument)*
55 ;
56
57 Argument:
58   name=ID
59 ;
60
61 CustomArguments:
62   arguments+=ArgumentOrObjectOrLocation (',' arguments+=ArgumentOrObjectOrLocation)*
63 ;
64
65 ArgumentOrObjectOrLocation:
66   {Arg}'arg' arg=[Argument]| {Objdec}'objdec' objdec=[ObjectDeclaration]| {Locdec}'locdec
       ' locdec=[LocationDeclaration]
67 ;
68
69 ConditionDeclaration:
70   'Condition' conditions=Conditions?
71 ;
72
73 Conditions:
74   conditions+=ActionCondition (',' conditions+=ActionCondition)*
75 ;
76
77 ActionCondition:
78   VarActionCondition | InternalFunction | BooleanExp
79 ;
80
81 VarActionCondition:
82   not='!'? argument=[Argument] '.' property=[Property]
83 ;
84
85 InternalFunction:
86   not='!'? internal_name=InternalName '(' arguments=CustomArguments? ')'
87   //not='!'? internal_name=InternalName '(' arguments=Arguments? ')'
88 ;
89
90 PropertyAssignment returns Assignment:
91   NormalAssignment | IncDecAssignment
92 ;
93
94 NormalAssignment returns Assignment:
95   (dec_name=ID '.')? assign_name=[Property] op=AssignOperator exp=MathExp
96 ;
97
98 AssignOperator:
99   {Eq} '=' | {PlusEq} '+=' | {MinusEq} '-=' | {MultEq} '*=' | {DivEq} '/='
100 ;
101
102 IncDecAssignment returns Assignment:
103   (dec_name=ID '.')? assign_name=[Property] op=IncDecOp
104 ;
105
106 IncDecOp:
107   {Inc} '++' | {Dec} '--'
108 ;
109
110 InternalName:
111   'isNeighbor' | 'isHere' | 'goTo'
112 ;
```

```
113 CustomFunctionName:
114    'delete' | 'create'
115 ;
116
117 EffectDeclaration:
118    'Effect' effects=Effects?
119 ;
120
121 Effects:
122    effects+=CustomEffects (',' effects+=CustomEffects)*
123 ;
124
125 CustomEffects:
126    PropertyAssignment | InternalFunction | CustomFunction
127 ;
128 CustomFunction:
129    internal_name=CustomFunctionName '(' arguments=Arguments? ')'
130 ;
131
132 WinningState:
133    'WinningState' conditions=WinningConditions
134 ;
135
136 WinningConditions:
137    cond+=WinningCondition (',' cond+=WinningCondition)*
138 ;
139
140 WinningCondition:
141    VarWinCondition | InternalFunction | BooleanExp
142 ;
143
144 VarWinCondition:
145    not='!'? var_name=ID '.' property=[Property]
146 ;
147
148 BooleanExp returns Proposition:
149    LogicExp (({And.left=current} '&&' | {Or.left=current} '||') right=LogicExp)*
150 ;
151
152 LogicExp returns Proposition:
153    Comparison | {BooleanValue} bool=BooleanValue
154 ;
155
156 BooleanValue:
157    'true' | 'false'
158 ;
159
160 Comparison returns Proposition:
161    {Comparison} left=MathExp operator=EqOp right=MathExp
162 ;
163
164 MathExp returns Expression:
165    Factor (({Add.left=current} '+' | {Sub.left=current} '−') right=Factor)*
166 ;
167
168 Factor returns Expression:
169    Prim (({Mult.left=current} '*' | {Div.left=current} '/') right=Prim)*
170 ;
171
172 Prim returns Expression:
173    {Number} value=INT | {Parenthesis} '(' exp=MathExp ')' | {Variable} var_prop=[Property]
        | {LocalVariable} var_local=ID '.' var_prop=[Property]
```

```
174  ;
175
176  EqOp:
177      '==' | '>=' | '<=' | '>' | '<'
178  ;
```

Listing 10: Grammar

A.1.2 Expression

```
1    def List<Expression> getVariables(Property p) {
2      switch p {
3        BoolExp: p.bool_exp.getBoolVars
4        NumberExp: p.math_exp.getMathVars
5        default: throw new Error("Invalid expression")
6      }
7    }
8
9    def List<Expression> getBoolVars(Proposition p) {
10     var list = new ArrayList<Expression>()
11     switch p {
12       And: {
13         list.addAll(p.left.getBoolVars)
14         list.addAll(p.right.getBoolVars)
15       }
16       Or: {
17         list.addAll(p.left.getBoolVars)
18         list.addAll(p.right.getBoolVars)
19       }
20       Comparison: {
21         list.addAll(p.left.getMathVars)
22         list.addAll(p.right.getMathVars)
23       }
24     }
25     list
26   }
27
28   def List<Expression> getMathVars(Expression e) {
29     var list = new ArrayList<Expression>()
30     switch e {
31       Add: {
32         list.addAll(e.left.getMathVars)
33         list.addAll(e.right.getMathVars)
34       }
35       Sub: {
36         list.addAll(e.left.getMathVars)
37         list.addAll(e.right.getMathVars)
38       }
39       Mult: {
40         list.addAll(e.left.getMathVars)
41         list.addAll(e.right.getMathVars)
42       }
43       Div: {
44         list.addAll(e.left.getMathVars)
45         list.addAll(e.right.getMathVars)
46       }
47       Parenthesis:
48         list.addAll(e.exp.getMathVars)
49       LocalVariable:
50         list.add(e)
51       Variable:
52         list.add(e)
53     }
```

```
54      list
55  }
```

Listing 11: Extension to get math expressions

B   Group rapport

B.1   The Domain

The motivation behind this project is to allow users with no previous extensive programming experience to create simple 2D top down games that can be displayed in a simple Graphical User Interface (GUI). This is possible by creating a static grid based system, where defined objects can be placed at X and Y coordinates. Furthermore, the user is also able to define custom conditions or make use of predefined functions to create the logic for the game. All this is achievable through an extendable framework that has been built from scratch.

B.2   Metamodel

The meta model as seen in figure 8 illustrates the most essential parts of the project. In order to explain the flow of the meta model a simplified version of the example game will be used.
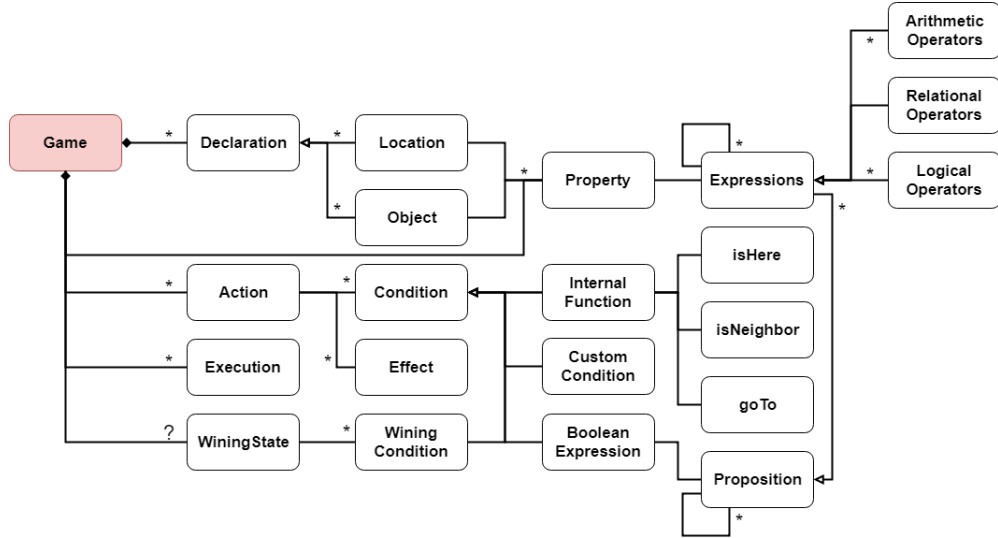

Figure 8: The meta model. Trivial parts omitted.

The model starts with a root Game AdventureQuest. A Game can have properties, objects or locations, actions, execute actions and an optional winning state.

An object is defined by using the Object keyword and providing a name to the object, as seen in the following Object Agent1 (1,1) with an initial position. Similarly a location can be declared. Objects and Locations can have properties such as truth value isAgent = true and number score = 0.

Actions are defined by using the Action keyword, specifying the name of the action and providing arguments to the action, Action Move (player, next). An

14

action consists of many conditions, including internal functions, as shown here Condition player.isAgent, player1.score < player2.score, isHere(player, key). An action must also contain effects, for example Effect player.keys++.

Notice that properties can also make use of multiple types of expressions, i.e. incrementation denoted by "++". Lastly an optional winning condition can be defined as shown, i.e. WinningState isHere(Agent1, Goal), Agent1.keys == 2. Actions can then be executed after the meta model is build, i.e. Move(Agent1, (0,0)). The extended game can be seen in listing 13.

B.3 The grammar

In this section there will be an elaboration on the different aspect of the grammar and how it is used when designing the metagame. The complete grammar is shown as xtext code in listing 14 in appendix B.8.1.

B.3.1 Composition and inheritance

An "Action" **has a** reference to an "ActionDeclaration", "ConditionDeclaration" and an "EffectDeclaration" rule (listing 14 line 33-34 appendix B.8.1). The reference is made by using "=" sign. In figure 12 appendix C.1 a class diagram of the composite relationship is shown.

An example of an inheritance relationship is the "Declaration" rule where "Object" and a "Location" are subclasses of a "Declaration" (listing 14 line 5-6 appendix B.8.1). A class diagram of the relationship is shown in figure 13 appendix C.1. It makes sense to use inheritance in this concrete situation since "Object" and "Location" **is a** specific instance of "Declaration". In section B.4 there will be an elaboration for generating code for these rules through dispatch methods.

B.3.2 The "returns" keyword

In some situation it can be convenient to return the same type for different rules. The BooleanExp (listing 14 line 133-134 appendix B.8.1), LogicExp (listing 14 line 137-138 appendix B.8.1) and Comparison (listing 14 line 145-146 appendix B.8.1) are examples of returning the same type, namely a "Proposition". As seen in figure 14 appendix C.1 the rule will produce an inheritance relationship.

B.3.3 Cross reference

With cross references the code is able to link objects together. A condition might have a "VarActionCondition" (variable action condition) which could be "player.isAgent" as seen in listing 13 line 13 section B.7.1. Looking at the grammar in listing 14 line 73-74 appendix B.8.1 there is a reference between an "Argument" and a "Property" on the "VarActionCondition" rule, indicated with "[""]". To get the cross reference to work properly it requires that the referenced rule has an "ID".

An advantage by using cross reference is that the editor will throw an error if the reference doesn't exists as seen in figure 9. Another way to test that the cross reference actual works is by clicking on the object together with control. It should then go to the referenced object.

```
    Game Labyrint
    number tcount = Agent1.score + Agent2.score

⊙ Object Agent1 (1,0) Agent2 (2,0)
        truth value isAgent = true
        number score = 0
        number x = Wall.test + 1
⊙ Location Wall (0,1) (2,3) (3,2)
        truth value isWall = true
        number test = Agent1.score + Agent2.score
    Object Goal (2,2)

⊙ Action Move (player,next)
         Condition player.isAgent, !next.isWall2, isNeighbor(player, next)
         Effect goTo(player, next), player.score++, player.x++
```

Figure 9: Cross reference error.

B.3.4 Left recursive, left associativity and precedence

The way the design removes left recursive grammar is by creating rules which
doesn't refer to the rule itself, as seen in listing 14 line 150 + 154 appendix
B.8.1.

By removing the left recursive calls, left associciativity needs to be handled.
Left associativity is handled by adding extra classes such as "Add.left=current",
"Sub.left=current" etc. in the rules. This results in left associciativity without
having left recursion. Figure 10 briefly illustrates the concept of left associativ-
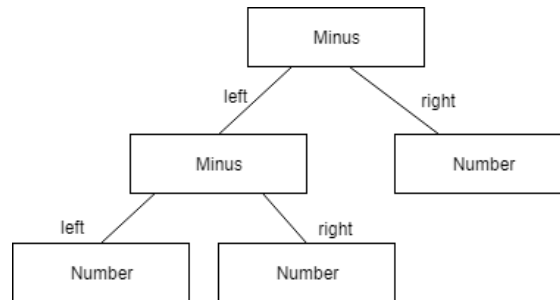ity.

Figure 10: Left associativity.

Operator precedence is handled by ensuring that operators are executed in
the correct order. Listing 14 line 149 + 153 appendix B.8.1, shows the rules
where operators with less precedence (+,-) are defined earlier than stronger
operators (*,/).

B.4 Xtend code and code generation

In this section the flow from the grammar to the generated code will be de-
scribed. Furthermore the generated file seen in listing 12 will be described.

Game meta model The metamodel is retrieved in the xtend generator as a
"Game" object as shown in figure 8. The generated file consists of a subclass
for each action, and a build and run method used to call an internal DSL

16

using the fluent interface and lastly a main method for execution. To retrieve all components; Global properties, Declarations, Actions, Winning State and Executions, from the "Game" object, each component is declared through a variable in xtext. This means composition is used in xtend to retrieve each of these components (listing 15 in Appendix B.8.2).

Generating actions   Each action is created as a subclass implementing the "IAction" interface, and has another interface injected in the constructor which can be used for calling internal functions implemented by the framework. The "execute" method, which should be called by the framework each time the action is run, takes a map and the metamodel as arguments. The map is for looking up the object or location object from the string parameter defined in the action. These string parameters are also being returned in the "getTypeList" method on the Action subclass for events to create the correct parameter map, that should be passed when executing the action. The parameters in the map are declared in new variables to be used by the conditions and effects. To be able to get the type of the parameter, type inference was implemented as seen below in listing 16 in Appendix B.8.2. The algorithm is based on which objects or locations had the local properties being used in the conditions or effects.

Now the action parameters has been declared and the conditions are generated to return a boolean value. The conditions can be of three types, "VarActionCondition", "InternalFunction" or "Proposition", which is determined through dispatch methods as seen in listing 17 in Appendix B.8.2. The methods are implemented to handle the condition hierarchy. All boolean values of the conditions are determined using conjunction if the effects should be performed. The effects are also generated using dispatch methods due to the effect hierarchy, which can be of type "InternalFunction" or "Assignment". The "Assignment" type is generated using the assignment operators defined in the grammar, and after each effect of type "Assignment" the code would generate the execution of the graph build from the fluent interface. The graph execution starts from the property being assigned, and updated all other properties dependent on the assigned property if any. An illustration of the graph execution can be seen in figure 15 appendix C.1.

Building the meta model   In the build method all "Object" and "Location" declarations with their properties are created along with the "WinningState". Properties can both be global fields or attached to objects or locations, therefore a prefix string is passed into the "generateProperty" method. To generate properties the type must be known, which can be either "bool", "int" or "var". For the "bool" and "int" type the logical or mathematical expression is displayed directly in java, but for the "var" type, the value can not just be displayed, because it is dependent on other properties properties, i.e. property a is dependent on property b in "number a = b + 1". Therefore we implement the expression as a callback, which can be executed by the framework whenever the dependent properties changes. To generate boolean and math expressions the generate methods took "Proposition" and "Expression" respectively as arguments as seen in listing 18 in Appendix B.8.2, making it possible to call itself recursively with left associativity. It is seen in the listing, that variables and local variables are displayed as a lookup in the parameter map to keep the ex-

pressions dynamic. A variable is typically a global variable on the game, i.e. "totalCount" and local variables are variables on objects, locations or actions, i.e. "Agent.isAgent".

The "WinningState" is also generated as a callback, which the framework should call after every performed action to determine if the player has won. This callback is build up the same way as the actions' "execute" method, where the conditions are generated and processed by conjunction to return whether the player has won the game.

Executing actions   At last the executions are generates as "Event" objects with a list of arguments, either name strings or positions, and an instance of the created action subclass. When the main method is run after generation and the project has a reference to the framework, the generated code will be coupled with the framework through inheritance using the Generation Gap design pattern as described in [Fow10]. This avoids having the user to create handwritten code in the generated code.

B.5   Generated Code for Adventure Quest game

```java
public class AdventureQuest extends FluentMachine {

  public class MoveAction implements IAction{
    private IInternalFunction _iInternalFunction;

    public MoveAction(IInternalFunction iInternalFunction){
       _iInternalFunction = iInternalFunction;
    }

    public MoveAction(){}

    @Override
    public boolean execute(Map<String, java.lang.Object> args, MachineMetaModel model) {
      boolean executionAllowed = true;
      Map<String, java.lang.Object> map = model.getExtendedStateVariables();
      MetaGameGraph graph = model.getGraph();
      Object player = (Object)args.get("player");
      Location next = (Location)args.get("next");
      executionAllowed &= (map.containsKey(player.getName()+".isAgent") && (boolean)map.
get(player.getName()+".isAgent")) || !map.containsKey(player.getName()+".isAgent");
      executionAllowed &= (map.containsKey(next.getName()+".isWall") && !(boolean)map.get
(next.getName()+".isWall")) || !map.containsKey(next.getName()+".isWall");
      executionAllowed &= _iInternalFunction.isNeighbor(player, next);
      if(executionAllowed) {
         _iInternalFunction.goTo(player, next);
      }
      return executionAllowed;
    }

    @Override
    public List<String> getTypeList() {
    List<String> types = new ArrayList<>();
    types.add("player");
    types.add("next");
    return types;
    }
  }

```

```java
37   public class PickUpKeyAction implements IAction{
38     ...
39   }
40
41   private MachineMetaModel _machineMetaModel;
42   private List<Event> events = new ArrayList<>();
43   private IAction action = new MoveAction(new FrameworkPredefinedInternalFunction());
44   private AdventureQuest(){
45     build();
46     run();
47   }
48
49   public void build(){
50     Map<String, java.lang.Object> map = metamodel.getExtendedStateVariables();
51     // Create object tree
52     global("AdventureQuest")
53       .object("Agent1",1,0).boolProperty("isAgent", true).intProperty("keys", 0)
54       .object("Agent2",1,1).boolProperty("isAgent", true).intProperty("keys", 0)
55       .object("Key1",0,0).boolProperty("isKey", true)
56       .object("Key2",3,1).boolProperty("isKey", true)
57       .location("Wall", new ArrayList<Position>()  {{ add(new Position(2,0)); add(new
    Position(2,1)); add(new Position(1,3)); add(new Position(0,3));}}).boolProperty("
    isWall", true)
58       .object("Goal",3,0)
59       .winningState(new IWinningCallback() {
60         @Override
61         public boolean execute(MachineMetaModel model, IInternalFunction
    _iInternalFunction) {
62           Map<String, java.lang.Object> map = model.getExtendedStateVariables();
63           boolean hasWon = true;
64           hasWon &= _iInternalFunction.isHere(model.getObject("Agent1"), model.getObject
    ("Goal"));
65           hasWon &= _iInternalFunction.isHere(model.getObject("Agent2"), model.getObject
    ("Goal"));
66           hasWon &= (((int)map.get("Agent1.keys")+(int)map.get("Agent2.keys")) == (4));
67           return hasWon;
68         }
69       });
70     // Create events
71     events.add(new Event(new ArrayList<java.lang.Object>() {{ add("Agent1"); add(new
    Position(0, 0)); }}, new MoveAction(new FrameworkPredefinedInternalFunction())));
72     ...
73
74   _machineMetaModel = metamodel;
75   }
76
77   public void run(){
78     IFramework framework = new GameFramework(new GameFrameworkFactory(_machineMetaModel,
    new FrameworkPredefinedInternalFunction()));
79     framework.run(events);
80   }
81
82   public static void main(String [] args) {
83     new AdventureQuest();
84   }
85 }
```

Listing 12: Generated code

B.6 Framework

The framework is the core engine for coupling the generated code together with the handwritten code[Fow10]. This code base consist of a game framework and

19

an internal DSL for creating a state machine through a fluent interface. The generated code, generates internal DSL code for a state machine together with the action classes, which then gets injected into the game framework. A detailed overview of the fluent machine class can be seen in figure 17 appendix 17.

A state machine design made sense, since a game has an initial state, inputs events which may or may not trigger some actions and new states are generated based on the input. The game framework is a facade which is responsible for executing the events and returning new states, which are passed on to a presentation layer. The overall code base is a result of an iterative process based on known software principles such as flexible and modifiable "qualities" as described in [BCK12] and [Chr10].

The meta model stores all updated information about the itself. Through the meta model all states, all objects, all events, all locations, execution graph etc. are achievable. A detailed overview of the meta model and the state machine executor can be seen in figure 16 appendix C.2.

B.6.1   Module viewpoint

The handwritten code base of the framework consist of 27 classes. In order to avoid showing all the classes in a class diagram, a package diagram will be used instead. As seen in figure 11, the diagram illustrates the dependencies between the different packages.

The generated java code builds a meta model through an internal DSL by inheriting the fluent interface. When the meta model is build, the game framework is instantiated with the meta model and a list of events to be processed. Last the game framework processes each event from the event list by calling the executor and then updates the GUI for each processed event.

A console output and simple gui of the game board has been created which can be seen in figure 18 and 19 appendix C.2.
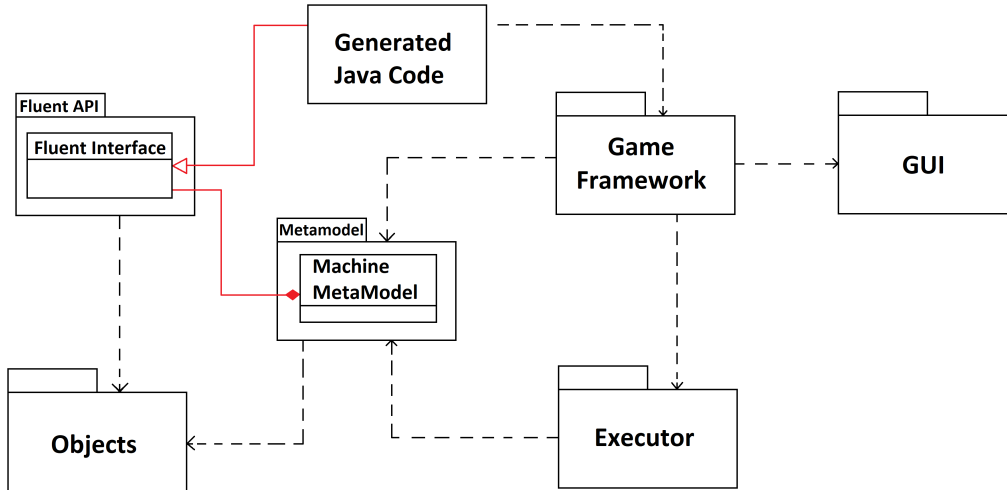

Figure 11: Package diagram mapping the functionality.

B.7 Example of games

B.7.1 Adventure Quest

The first game in listing 13 is a game where agents should collaborate on picking up a certain amount of keys before winning the game.

The first lines from 1-20 is a declaration of the game which consists of 2 agents, 2 keys, 4 walls, 1 goal, 2 actions and a winning state. On the Agent objects, position and properties are declared such as "isAgent" and "keys". On the Key objects, position and a property "isKey" is declared and etc.

The "Move" action defines what conditions should be fulfilled before an agent is able to move. The condition states that the "player" should be an agent and the location where the player want to move must not be a wall. Finally it checks if the location is next to the player through a framework internal function. If the conditions are fulfilled an effect will be executed which is the actual move. The "goTo" is also a framework internal function.

The "PickUpKey" action defines what should be fulfilled when a player is able to pick up keys. The condition state that a player should be an agent, the "key" should be a key and that the player should be placed on a key. If these conditions are fulfilled an effect will be executed which is incrementing the amount of keys with 1 on the player.

The "WinningState" declares that both agents should be located on the goal and that their sum of keys should be equal to 4.

The last part from line 22-44 actions are executed for both Agents for winning the game. The actions shows that the agents only are winning the game when their sum of keys are 4.

```
1  Game AdventureQuest
2
3  Object Agent1 (1,0) Agent2 (1,1)
4    truth value isAgent = true
5    number keys = 0
6  Object Key1 (0,0) Key2 (3,1)
7    truth value isKey = true
8  Location Wall (2,0) (2,1) (1,3) (0,3)
9    truth value isWall = true
10 Object Goal (3,0)
11
12 Action Move (player, next)
13   Condition player.isAgent, !next.isWall, isNeighbor(player, next)
14   Effect goTo(player, next)
15
16 Action PickUpKey (player, key)
17   Condition player.isAgent, key.isKey, isHere(player, key)
18   Effect player.keys++
19
20 WinningState isHere(Agent1, Goal), isHere(Agent2, Goal), Agent1.keys + Agent2.keys == 4
21
22 Move(Agent1, (0,0))
23 PickUpKey(Agent1, Key1) // Agent1 collects one key
24 Move(Agent1, (0,1))
25 Move(Agent2, (1,0))
26 Move(Agent2, (0,0))
27 PickUpKey(Agent2, Key1) // Agent2 collects another key
28 Move(Agent1, (0,2))
29 Move(Agent1, (1,2))
30 Move(Agent1, (2,2))
31 Move(Agent1, (3,2))
```

```
32  Move(Agent1, (3,1))
33  Move(Agent1, (3,0)) // Agent1 reaches goal - 2 keys in total
34  Move(Agent2, (0,1))
35  Move(Agent2, (1,1))
36  Move(Agent2, (1,2))
37  Move(Agent2, (2,2))
38  Move(Agent2, (3,2))
39  Move(Agent2, (3,1))
40  PickUpKey(Agent2, Key2) // Agent2 collects another key
41  Move(Agent2, (3,0)) // Agent2 reaches goal - 3 keys in total
42  Move(Agent1, (3,1))
43  PickUpKey(Agent1, Key2) // Agent1 collects another key
44  Move(Agent1, (3,0)) // Agent1 reaches goal - 4 keys in total = Winning State
```
Listing 13: Game AdventureQuest

B.7.2 Additional games sketches

Furtheremore we have three additional game sketches; "Multi agents", "Racing track" and "Labyrinth".

Multi agents is a game where multiple agents needs to push boxes that have the same colour as themselves, to a goal.

Racing track is a game where cars needs to run a certain number of user defined laps. While driving the race cars need to pass a certain amount of check points, before they are able to win the race.

Labyrinth is a game where an agent needs to find a path through a labyrinth of walls. To win the game, the agent is only allowed to take a certain amount of steps before reaching the goal.

All game sketches are added to the code base.

B.8   Listings

B.8.1   Grammar

```
1  Game:
2    'Game' name=ID fields+=Property* declarations+=Declaration* actions+=Action*
       winningState=WinningState? executions+=Execution*
3  ;
4
5  Declaration:
6      Object | Location
7  ;
8
9  Location:
10   'Location' declarations+=LocationDeclaration+ properties+=Property*
11  ;
12
13  LocationDeclaration:
14    name=ID coordinates+=Coordinates+
15  ;
16
17  Object:
18   'Object' declarations+=ObjectDeclaration+ properties+=Property*
19  ;
20
21  ObjectDeclaration:
22    name=ID coordinates=Coordinates
23  ;
```

```
24
25  Coordinates :
26    '('x=INT ','y=INT ')'
27  ;
28
29  Property :
30    {BoolExp} 'truth ' 'value ' name=ID '=' bool_exp=BooleanExp | {NumberExp} 'number ' name=
        ID '=' math_exp=MathExp
31  ;
32
33  Action :
34    declaration=ActionDeclaration condition=ConditionDeclaration effect=EffectDeclaration
35  ;
36
37  ActionDeclaration :
38    'Action ' name=ID '(' args=Arguments? ')'
39  ;
40
41  Execution :
42     action_name=[ActionDeclaration ]    '(' executionArgs=ExecutionArg? ')'
43  ;
44
45  ExecutionArg :
46    executionArgs+=ExecutionDeclaration (','executionArgs+=ExecutionDeclaration)∗
47  ;
48
49  ExecutionDeclaration :
50    Argument | Coordinates
51  ;
52
53  Arguments :
54    arguments+=Argument (',' arguments+=Argument)∗
55  ;
56
57  Argument :
58    name=ID
59  ;
60
61  ConditionDeclaration :
62    'Condition ' conditions=Conditions?
63  ;
64
65  Conditions :
66    conditions+=ActionCondition (',' conditions+=ActionCondition)∗
67  ;
68
69  ActionCondition :
70    VarActionCondition | InternalFunction | BooleanExp
71  ;
72
73  VarActionCondition :
74    not='!'? argument=[Argument] '.' property=[Property]
75  ;
76
77  InternalFunction :
78    not='!'? internal_name=InternalName '(' arguments=Arguments? ')'
79  ;
80
81  PropertyAssignment returns Assignment :
82    NormalAssignment | IncDecAssignment
83  ;
84
```

```
85  NormalAssignment returns Assignment :
86    assign_name=[Property] op=AssignOperator exp=MathExp
87  ;
88
89  AssignOperator :
90    {Eq} '=' | {PlusEq} '+=' | {MinusEq} '-=' | {MultEq} '*=' | {DivEq} '/='
91  ;
92
93  IncDecAssignment returns Assignment :
94    (dec_name=ID '.')? assign_name=[Property] op=IncDecOp
95  ;
96
97  IncDecOp :
98    {Inc} '++' | {Dec} '--'
99  ;
100
101 InternalName :
102   'isNeighbor' | 'isHere' | 'goTo'
103 ;
104
105 EffectDeclaration :
106   'Effect' effects=Effects?
107 ;
108
109 Effects :
110   effects+=CustomEffects (',' effects+=CustomEffects)*
111 ;
112
113 CustomEffects :
114   PropertyAssignment | InternalFunction
115 ;
116
117 WinningState :
118   'WinningState' conditions=WinningConditions
119 ;
120
121 WinningConditions :
122   cond+=WinningCondition (',' cond+=WinningCondition)*
123 ;
124
125 WinningCondition :
126   VarWinCondition | InternalFunction | BooleanExp
127 ;
128
129 VarWinCondition :
130   not='!'? var_name=ID '.' property=[Property]
131 ;
132
133 BooleanExp returns Proposition :
134   LogicExp (({And.left=current} '&&' | {Or.left=current} '||') right=LogicExp)*
135 ;
136
137 LogicExp returns Proposition :
138   Comparison | {BooleanValue} bool=BooleanValue
139 ;
140
141 BooleanValue :
142   'true' | 'false'
143 ;
144
145 Comparison returns Proposition :
146   {Comparison} left=MathExp operator=EqOp right=MathExp
```

```
147 ;
148
149 MathExp returns Expression:
150    Factor (({Add.left=current} '+' | {Sub.left=current} '−') right=Factor)∗
151 ;
152
153 Factor returns Expression:
154    Prim (({Mult.left=current} '∗' | {Div.left=current} '/') right=Prim)∗
155 ;
156
157 Prim returns Expression:
158    {Number} value=INT | {Parenthesis} '(' exp=MathExp ')' | {Variable} var_prop=[Property]
         | {LocalVariable} var_local=ID '.' var_prop=[Property]
159 ;
160
161 EqOp:
162    '==' | '>=' | '<=' | '>' | '<'
163 ;
```

Listing 14: Grammar

B.8.2 Xtend code

```
1    <<FOR f: game.fields >><<f.generateProperty("")>><<ENDFOR>>
2    <<FOR d: game.declarations >><<d.generateDeclaration >><<ENDFOR>>
```

Listing 15: Composition

```
1    def String getArgumentType(Argument arg, Action action, List<Declaration> declarations)
         {
2    // Find argument properties in action
3      var targetProperty = ""
4      for (c:action.condition.conditions?.conditions) {
5        if (c instanceof VarActionCondition) {
6          var custom = c as VarActionCondition
7          if (custom.argument.name.equals(arg.name)) {
8            targetProperty = custom.property.name
9          }
10       }
11     }
12     if (targetProperty == "") throw new Exception("Type inference error..")
13     // Find type by properties in declarations
14     var type = ""
15     for (d:declarations) {
16       var currentType = d.getType(targetProperty)
17       if (!currentType.equals("")) {
18         type = currentType
19       }
20     }
21     if (type.equals("")) throw new Exception("Type inference error..")
22     type
23
24   }
```

Listing 16: Type inference

```
1 def dispatch CharSequence generateCondition(VarActionCondition c)
2     ...
3
4 def dispatch CharSequence generateCondition(InternalFunction f)
5     ...
6
```

```
7 def dispatch CharSequence generateCondition(Proposition p) {
8    p.generateBoolExp
9 }
```

Listing 17: Dispatch methods

```
1 def String generateMathExp(Expression exp) {
2     switch exp {
3       Add: exp.left.generateMathExp+"+"+exp.right.generateMathExp
4       Sub: exp.left.generateMathExp+"-"+exp.right.generateMathExp
5       Mult: exp.left.generateMathExp+"*"+exp.right.generateMathExp
6       Div: exp.left.generateMathExp+"/"+exp.right.generateMathExp
7         Number: Integer.toString(exp.value)
8         LocalVariable: exp.var_prop.getPropertyType + "map.get(\""+exp.var_local +"."+
      exp.var_prop.name+"\")"
9         Variable: exp.var_prop.getPropertyType + "map.get(\""+exp.var_prop.name+"\")"
10        default: throw new Error("Invalid Math Expression")
11    }
12 }
```

Listing 18: Math expressions

C   Figures

C.1   Xtext Semantics
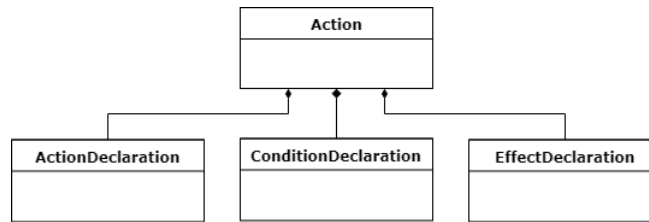
Figure 12: Class diagram of the composition relationship.
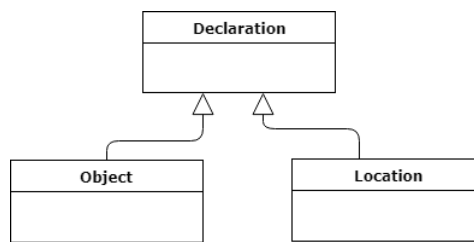
Figure 13: Class diagram of the inheritance relationship.
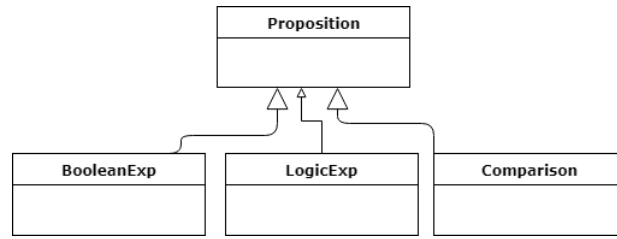
Figure 14: Class diagram of the inheritance relationship when using "returns".
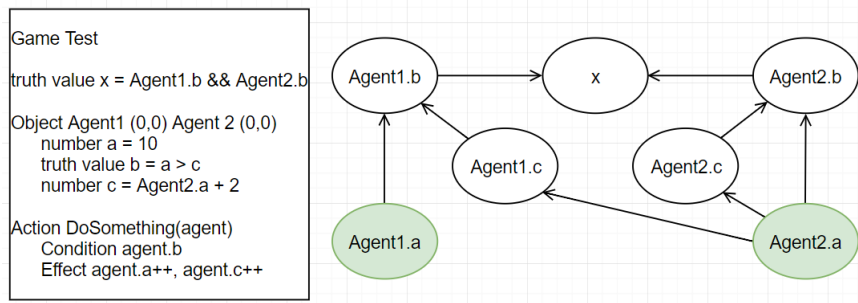

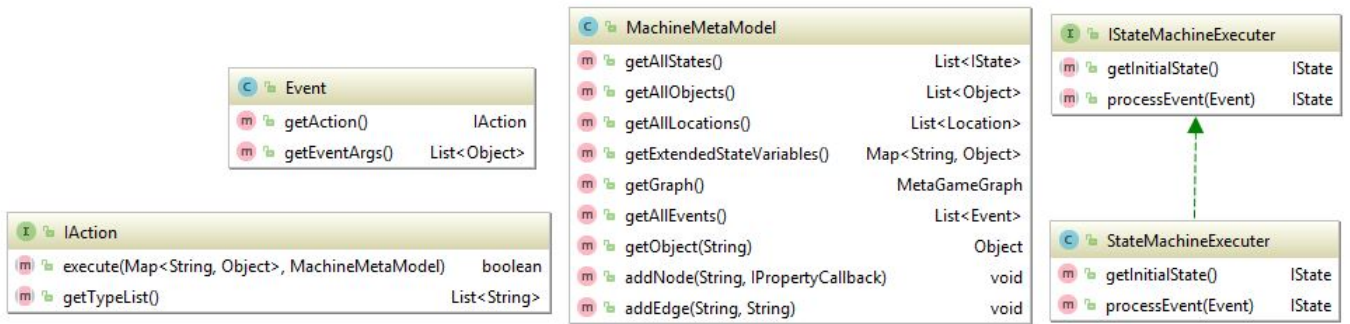Figure 15: Example of an execution graph

C.2   Gameframework

Figure 16: MachineMetaModel and StateMachineExecuter classes.
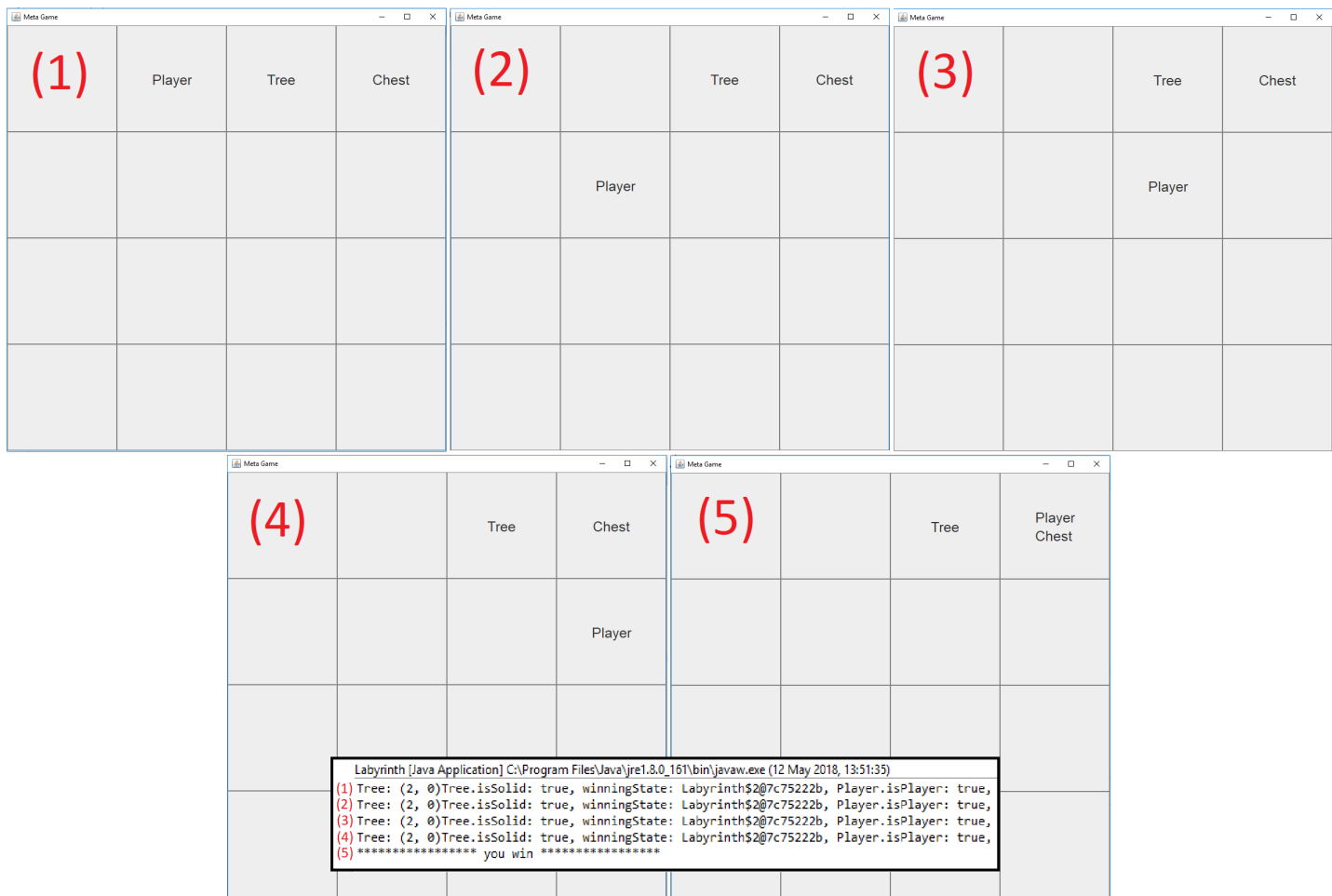
Figure 17: The fluent machine class.

Figure 18: Simple console output.

Figure 19: Example GUI for a simple game.