A Case Study of the Component and Connector Modeling Language EmbeddedMontiArc

Philipp Haller, Malte Heithoff

Supervised by: Michael von Wenckstern and Bernhard Rumpe
Software Engineering, RWTH Aachen University

Abstract

(Abstract by Philipp Haller) The magnitude and quantity of software projects rises constantly, as software development needs spread among scientific and technical disciplines. Domain Specific Languages (DSLs) show to provide solutions for specialized contexts. EmbeddedMontiArc is a DSL for cyber physical systems. This paper represents a case-study, evaluating the ease of use and reusability of EmbeddedMontiArc for reactive systems by presenting models for the games Pacman and Supermario. Games are highly reactive systems were entities controlled by the player react to a changing environment and try to reach goals, thus can provide a good testing ground for actual systems. From the models presented it is concluded that EmbeddedMontiArc is suitable for cyber-physical systems, but still not flawless.

Keywords: EmbeddedMontiArc, Component & Connector Models, Case Study, Supermario, Pacman

☆

1. Introduction (by Philipp Haller)

The magnitude and quantity of software projects rises constantly, as software development needs spread among scientific and technical disciplines. Since not all languages are suitable for all occasions and others may provide too much features to be efficient for a specific purpose, Domain Specific Languages (DSLs) are developed. DSLs are languages tailored specifically to a certain objective. EmbeddedMontiArc, a specific DSL for cyber-physical systems is evaluated in this paper. It will be introduced in more detail in section 2 together with the used tools. This section forms a general introduction and will present the research questions. Thereafter the approach will be presented in section 3. Section 4 depicts the simulator integration and the developed models. In section 5 the evaluation is presented, concluded by a conclusion in section 6.

In general most problems can be sorted into two categories. The first being data based problems, where huge amounts of data are processed and no hard real time capabilities are necessary. An example for such a problem is Google's or Amazon's search system. The other problem category consists of reactive systems which operate on very little data and must return output with hard time constraints. In this paper EmbeddedMontiArc is evaluated towards its capabilities for the second category. The language is well tested on the autopilot project of a self driving car (see [1]), but has few other running examples. The following research questions were formulated to specify evaluation topics:

- RQ1: Is EmbeddedMontiArc suitable reactive systems in domains other than the automotive industry?
- RQ2: Is it possible to integrate other simulators in a recent amount of work?
- RQ3: What kind of background knowledge is needed to model C&C in EmbeddedMontiArc?
- RQ4: What features are good and what are not suited?

To answer these research questions two groups are formed who develop different models in EmbeddedMontiArc and share their experience while doing so. To ensure a similar experience to real reactive cyber physical systems, two games were selected. Games were selected, because most games are real-time problems with a changing environment and limited inputs, while requiring immediate responses. The games chosen for this paper are Pacman and Supermario. Both games are 2D arcade games where a figure is controlled by a player in a setting where some types of enemy entities exist. In the case of Pacman the level is completely visible and enemies consist of four ghosts roaming the level. The level is failed once the ghosts touch the player. Goal of the game is to collect or "eat" all dots in the level. Supermario is a side-scrolling platform game were the level is revealed as the player progresses. Main goal of Supermario is to bring the player figure all the way through to the end of the level, while either evading or defeating the different enemy types. The players progress is rated via a scoring system, where enemy defeats and collectibles are assessed. Goal for both models developed in this paper is to solve a level in their respective game.

The finished models can be observed playing Pacman and Supermario autonomouosly on the websites
`https://embeddedmotiarc.github.io/SuperMario/Pacman/`[2]
and
`https://embeddedmontiarc.github.io/SuperMario/supermario/simulation.html`[3].

2. Context (by Philipp Haller)

The following section consists of three parts. The first one is a brief introduction to C&C models. The tools used for this study follow up second. Lastly, the used case study method is presented.

2.1. C & C models

In the following a short introduction in Connector and Component (C&C) model based software development is given. C&C modeling divides a task into Components and Connectors.

A *Component* represents a computation. It has predefined inputs and outputs, where the output data is obtained by some kind of mathematical transformation of the input data. A *Connector* represents interaction mechanisms by connecting outputs with inputs. By making this division, the paradigm ensures modularity and therefore re-usability. It can be used for modeling software with high demands for testing and verification such as software for self-driving vehicles [4][5]. Another benefit is that a graphical representation is always possible and more efficiently obtainable compared to other text based development, especially non model driven development. The structure of C&C models also benefits code generation techniques in order to transform models into source code for various target systems. Well established examples of C&C modeling and development are SysML[6], AADL[7], Simulink[8] and Labview[9]. The latter two are used in the automotive domain to model behaviour of Electronic Control Units (ECUs) and test their functionality.

2.2. MontiCore and EmbeddedMontiArc

MontiCore [10], MontiCAR [11] and EmbeddedMontiArc [12] are tools developed by the Chair of Software Engineering of RWTH Aachen University[13]. *MontiCore* is a language workbench intended for agile and model-driven software development. Its primary objective is to enable efficient development of Domain Specific Languages (DSLs) which enhance the development process for Domain Experts. *MontiCAR* is a composition of such DSLs, used as an language set for Cyber-Physical Systems [14]. Figure 1 shows the DSLs which are part of MontiCar and their respective connections. The components directly used in this studies implementation are EmbeddedMontiArc, EmbeddedMontiArcMath

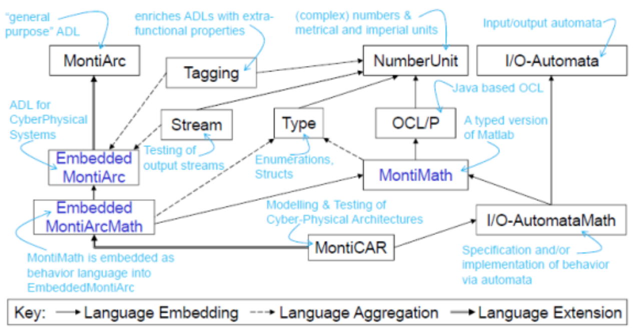
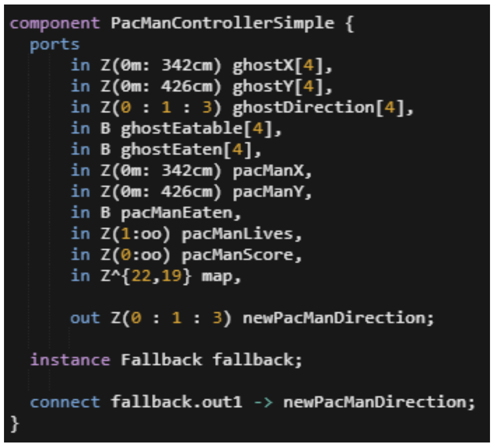

"general purpose" ADL
enriches ADLs with extra-functional properties
(complex) numbers & metrical and imperial units
Input/output automata
MontiArc
Tagging
NumberUnit
I/O-Automata
Java based OCL
ADL for CyberPhysical Systems
Stream
Type
OCL/P
A typed version of Matlab
Embedded MontiArc
Testing of output streams
Enumerations, Structs
MontiMath
Embedded MontiArcMath
Modelling & Testing of Cyber-Physical Architectures
I/O-AutomataMath
MontiCAR
MontiMath is embedded as behavior language into EmbeddedMontiArc
Specification and/or implementation of behavior via automata
Key:
Language Embedding
Language Aggregation
Language Extension

Figure 1: Composition of MontiCAR language family[14]

and Stream. *EmbeddedMontiArc* represents the core language of MontiCar. It implements a C&C DSL which can be used to write C&C models, verify, test and deploy them to another architecture. Due to its modularity different simulators and Stream tests can be integrated. See the chapter modeling for more information. Figure 2 depicts a usage of the EmbeddedMontiArc DSL.

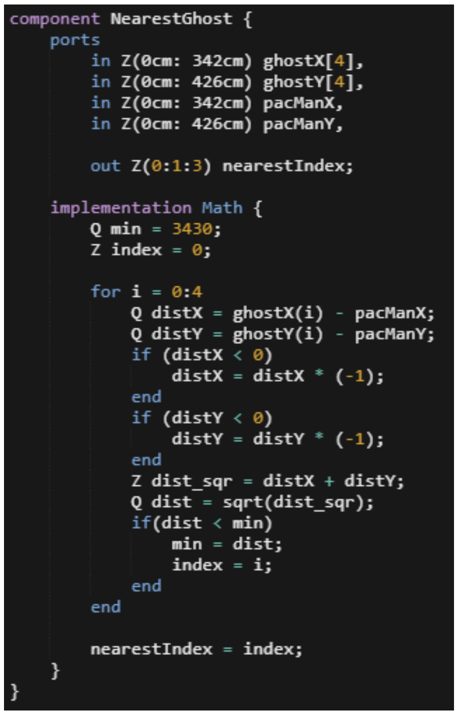


```
component PacManControllerSimple {
  ports
      in Z(0m: 342cm) ghostX[4],
      in Z(0m: 426cm) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],
      in B ghostEaten[4],
      in Z(0m: 342cm) pacManX,
      in Z(0m: 426cm) pacManY,
      in B pacManEaten,
      in Z(1:oo) pacManLives,
      in Z(0:oo) pacManScore,
      in Z^{22,19} map,

      out Z(0 : 1 : 3) newPacManDirection;

  instance Fallback fallback;

  connect fallback.out1 -> newPacManDirection;
}
```

Figure 2: Example Component with Connectors

EmbeddedMontiArcMath is a DSL for implementing mathematical expressions, thus used for transforming the input values of a Component into its output values. It is also able to declare other variables than the defined inputs and logical structures like if-statements and loops. Example usage of EmbeddedMontiArcMath is shown in figure 3.

The *Stream* DSL allows to implement test cases by defining the expected output values for a given input. Multiple values can be tested in one Stream test, as shown in figure 4 which shows an example stream test for a sum function. Thy syntax of this DSL holds the following items:

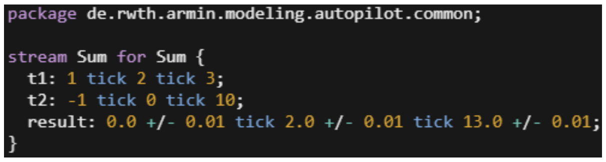


```
component NearestGhost {
    ports
        in Z(0cm: 342cm) ghostX[4],
        in Z(0cm: 426cm) ghostY[4],
        in Z(0cm: 342cm) pacManX,
        in Z(0cm: 426cm) pacManY,

        out Z(0:1:3) nearestIndex;

    implementation Math {
        Q min = 3430;
        Z index = 0;

        for i = 0:4
            Q distX = ghostX(i) - pacManX;
            Q distY = ghostY(i) - pacManY;
            if (distX < 0)
                distX = distX * (-1);
            end
            if (distY < 0)
                distY = distY * (-1);
            end
            Z dist_sqr = distX + distY;
            Q dist = sqrt(dist_sqr);
            if(dist < min)
                min = dist;
                index = i;
            end
        end

        nearestIndex = index;
    }
}
```

Figure 3: Example EmbeddedMontiArcMath implementation

- The package which also holds the component (de.rwth...)
- The test's name (Sum)
- The name of the component to test (Sum)
- Values for each input port (t1 and t2)
- A line with the values for at least one output port (result)

The values for one port are separated by ticks. Each tick stands for one execution cycle. This way a component can be tested over several cycles which gets important if the component's behavior is dependent on previous execution cycles. In addition the +/- allows inaccuracy in the results.



```
package de.rwth.armin.modeling.autopilot.common;

stream Sum for Sum {
  t1: 1 tick 2 tick 3;
  t2: -1 tick 0 tick 10;
  result: 0.0 +/- 0.01 tick 2.0 +/- 0.01 tick 13.0 +/- 0.01;
}
```

Figure 4: Example Stream implementation]

2.3. Performing a Case Study in Software Engineering

This study roughly follows the guidelines stated by Runeson and Hoest [15] by presenting the objective, the specific case, method and acquiring both quantitative and qualitative data. Quantitative data is acquired by asking a set of predefined questioned and answering them on a scale from 1 to 10. The qualitative data is obtained via requiring subjects to formalize how they gave the quantitative rating. The quantitative data is analyzed by calculating the mean of each question, and the quantitative by summarizing the subject's writings.

3. Approach

To address **RQ1** and **RQ3** two groups were assigned the task to model a controller for Pacman and Supermario respectively and interview the results afterwards. The first group (Pacman) consists of a subject who is familiar with EmbeddedMontiArc and the second group (Supermario) consists a subjects who have no experience with EmbeddedMontiArc. These groups were selected random among the students of a computer science seminar.

3.1. Stream Testing (by Heithoff)

EmbeddedMontiArc comes along with stream tests in order to check a component against a condition as stated in the previous chapter. We can use those tests to define the conditions the controllers need to fulfill. Those conditions are taken from use cases scenarios. For Pacman the most general acceptance test would be to never let the Pacman die. Due to the fact that stream tests cannot be defined unlimited and that this test might be hard to fulfill the following deterministic tests for Pacman and Supermario were defined.

3.1.1. Pacman (by Heithoff)

The tests are taken from use case scenarios as stated before. In this section the process of deriving the stream test from a scenario is presented once and then a few conditions are framed.

Deriving a Stream Test
In fig. 5 a scenario is shown where the only option for Pacman is to flee to the left in order to not collide with the pink and blue ghost. The values of the ghosts and Pacman are partially listed in listing 1. Together with the remaining values this concludes to the stream test shown below 2.

Some other tests
To formulate just some tests, here are a few examples:

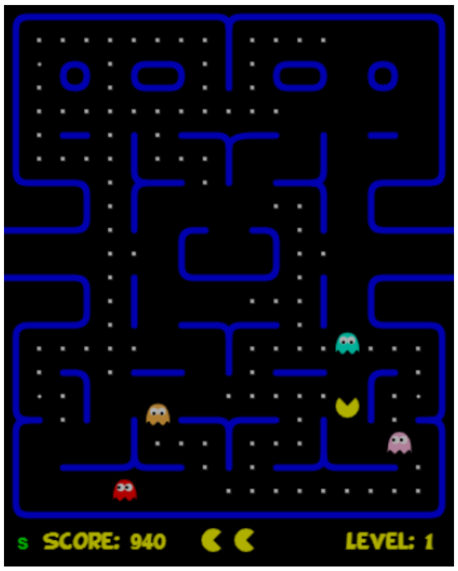
- If Pacman is located at an intersection and ghosts are coming from two sides, Pacman should walk to a safe path.
- If Pacman is located at an intersection and ghosts are at the top path and are all eatable, Pacman should walk this path.

Listing 1: Values for the stream test

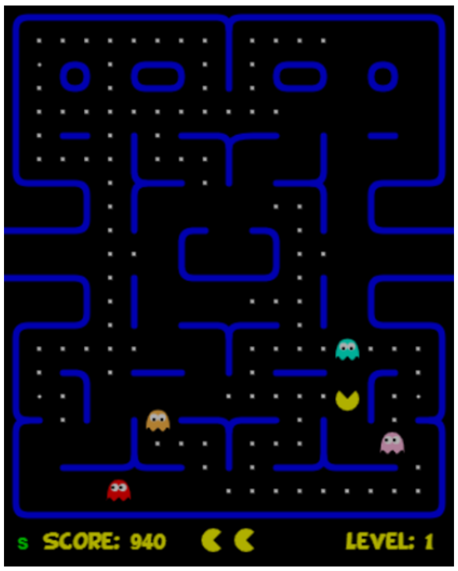
```
(a)
  Pacman: (15m, 17.2m)
  Pink Ghost: (17m, 19m)
  BlueGhost: (15m, 14.8m)
  newDir: 0
(b)
  Pacman: (15m, 17m)
  Pink Ghost: (16.8m, 19m)
  BlueGhost: (15m, 15m)
  newDir: 0
(c)
  Pacman: (14.8m, 17m)
  Pink Ghost: (16.6m, 19m)
  BlueGhost: (15m, 15.2m)
  newDir: 2
(d)
  Pacman: (14.6m, 17m)
  Pink Ghost: (16.4m, 19m)
  BlueGhost: (15m, 15.4m)
  newDir: 2
```

Listing 2: Stream test for the scenario above

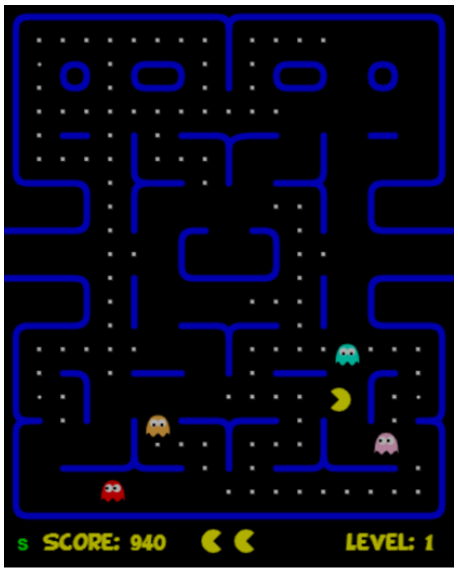
```
package de.rwth.Pacman;
stream Test1 for PacmanWrapper {
  ghostX: [5.4m,15m,17m,7m] tick [5.2m,15m, ...
  ghostY: [21m,14.8m,19m,17.2m] tick [21m,15m, ...
  ghostDirection: [2,1,2,1] tick [2,1,2,1] tick ...
  ghostEtable: [false, false, false, false] tick ...
  ghostEaten: [false, false, false, false] tick ...
  PacmanX: 15m tick 15m tick 14.8m tick 14.6m;
  PacmanY: 17.2m tick 17m tick 17m tick 17m;
  PacmanEaten: false tick false tick false tick false;
  PacmanLives: 3 tick 3 tick 3 tick 3;
  PacmanScore: 0 tick 0 tick 0 tick 0;
  map: [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0; ...
  newPacmanDirection: 0 tick 0 tick 2 tick 2;
}
```



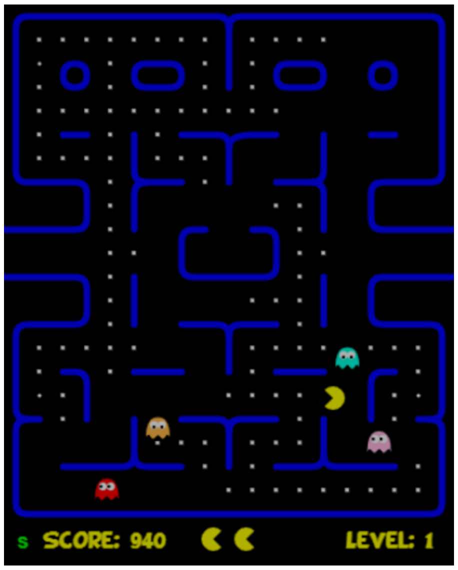
(a)



(b)



(c)



(d)

Figure 5: Pacman has to move left to avoid colliding with the ghosts

- If Pacman is located at an intersection and there are ghosts from 3 directions and in the other direction there is a ghost facing away from Pacman, Pacman should walk this direction.

- If there are no ghosts nearby, Pacman should walk the direction with the largest biscuit/coin value.

Those scenarios can be tested easily within a few ticks via stream testing.

3.1.2. Supermario (by Philipp Haller)

The goal for the Supermario model is to solve a level successfully. The first level was chosen since it provides a diverse environment with different enemy types and obstacles, while not being too skill intensive to solve. Prior to modeling some assumptions were made to fulfill time and complexity constraints. Only a fixed number of enemies and obstacles in the path of the player are considered in order to ensure a static input size. For this number, five has proved to be sufficient for the first level and the implemented strategy. There are rarely more than 3 enemies in scene. For the same reason only the next hole in the ground is considered. In order to develop the model, different situations were assessed and according tests derived. Both the scenarios which a Supermario model has to master and the derived tests are listed below.

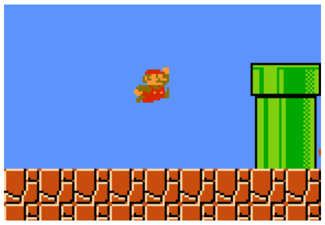
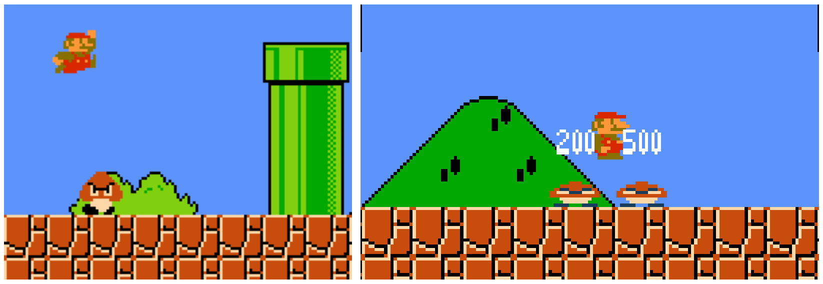


Figure 6: Mario has to jump and move right to overcome the obstacle

Figure 6 depicts the player next to an obstacle. In order to jump over it he has to move right and jump at the same time. He needs to keep jumping until he is higher than the obstacle.



(a) Mario evades a enemy by jumping (b) Mario defeats enemies by landing on them

Figure 7: Mario has to jump over/to enemies

Figure 7 shows two situations. In the first one, mario jumps to evade an enemy. The second depicts him landing on top of enemies to kill them.

In Figure 8 the player is seen standing next to holes in the ground. In the first picture he is on the ground level, in the second he is standing on an obstacle.

The stream tests derived from the scenarios are introduced in the following. If a enemy gets closer than 80 pixels (two blocks) and is on the same height as the player, the player has to jump in order to evade the enemy (listing 60). The units for the EnemyDistX and EnemyDistY values are pixels, while the velocities are given in pixels per time frame. The output values are of type boolean.

The stream in listing 61 covers the case when the player is above enemies and shall drop on them while he is above.

If there is no enemy near the player, the enemy watcher object shall give no jump advice (listing 62).

If a obstacle is in front of the player, he shall jump until he has passed it(listing 63). The distances are given in pixels, and the obstacle in this text is of 70px height.

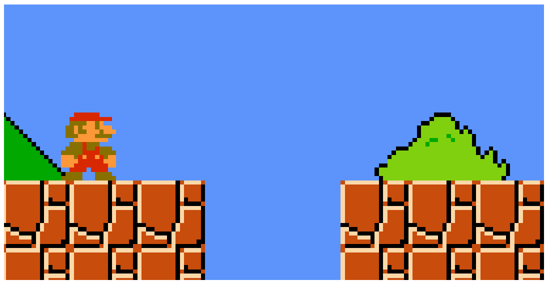
In listing 64 the stream test for jumping over holes is given. In this case, the player shall start jumping close to the hole and only stop once he is over.

3.2. Preparations (by Haller and Heithoff)

The code of the Pacman emulator [16] and Supermario emulator [17] is available in HTML5 and JavaScript. C&C-Components in EmbeddedMontiArc can be translated to C++ code and then to a web assembly [18] which uses JavaScript. This JavaScript file can be given inputs according to the component and calculates the outputs on execution. To combine these two files, there is an additional interface needed to extract the information for the inputs out of the emulator and then give the calculated outputs into the emulator. For the purpose of implementing the controllers the subjects were assigned to use the EmbeddedMontiArcStudio. EmbeddedMontiArcStudioV1.6.2 did neither

Listing 3: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_Evade for EnemyWatcher {
    EnemyDistX: 200 tick 100 tick 75;
    EnemyDistY: 0 tick 0 tick 0;
    EnemyVelocityX: −10 tick −10 tick −10;
    EnemyVelocityY: 0 tick 0 tick 0;

    movesTowardsPlayer: 1 tick 1 tick 1;
    inJumpRange: 0 tick 0 tick 1;
}
```


(a) Mario and a hole in the ground

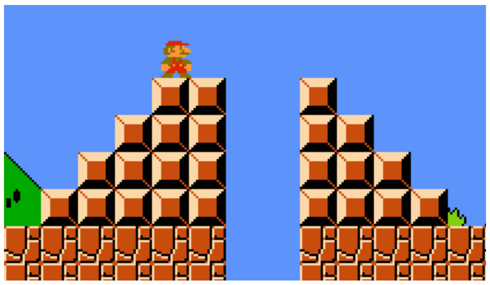

(b) Mario and a hole with obstacles

Figure 8: Mario has to jump over a hole

Listing 4: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_FromAbove for EnemyWatcher {
    EnemyDistX: 200 tick 100 tick 5;
    EnemyDistY: 128 tick 128 tick 32;
    EnemyVelocityX: −10 tick −10 tick −10;
    EnemyVelocityY: 0 tick 0 tick 0;

    movesTowardsPlayer: 1 tick 1 tick 1;
    inJumpRange: 0 tick 0 tick 0;

}
```

Listing 5: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_FromAbove for EnemyWatcher {
    EnemyDistX: −1 tick;
    EnemyDistY: −1 tick;
    EnemyVelocityX: 0 tick;
    EnemyVelocityY: 0 tick;

    movesTowardsPlayer: 0 tick;
    inJumpRange: 0 tick;
}
```

Listing 6: Obstacle watcher stream test

```
package de.rwth.supermario.haller.environment;
stream Env_ObstacleWatcher for ObstacleWatcher {
  ObstacleDistX: 200 tick 100 tick 75 tick 50 tick 25 tick 0;
  ObstacleDistX: 0 tick 0 tick 0 tick 25 tick 50 tick 75;

  inJumpRange: 0 tick 0 tick 1 tick 1 tick 1 tick 0;
}
```

Listing 7: Hole watcher stream test

```
package de.rwth.supermario.haller.environment;
stream Env_ObstacleWatcher for ObstacleWatcher {
    holeDistance: 200 tick 100 tick 10 tick 0 tick 1200;

    inJumpRange: 0 tick 0 tick 1 tick 1 tick 0;
}
```

support a simulator for Pacman nor a simulator for Supermario. So an additional step to answer RQ2 *Is it possible to integrate other simulators in a recent amount of work* it for the groups to integrate the simulators into the EmbeddedMontiArcStudio.

In order to be able to do so, group Pacman is instructed by an expert (Jean-Marc) which files need modification and what to add. After that this group instructed the second group the same way.

4. Case Study Execution

In this chapter the case study execution is described. First, the necessary steps for integrating a new Simulator into the IDE are shown. In the second part the modeling of the controllers for Pacman and Supermario are discussed.

4.1. Integration of Simulator into IDE (Introduction by Philipp Haller)

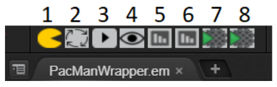
As the participants of the use-case study were divided into two groups, the first group dealt with the IDE integration of the Pacman simulator after being instructed by an EMA professional. After successful integration this first group wrote a step-by-step instruction list. The second group used this list to integrate the Supermario simulator into the IDE. Details are given in the following.

4.1.1. Integration at the example of Pacman (by Malte Heithoff)

To integrate a simulator into the EmbeddedMontiArcStudio several steps were necessary. In figure 9 you can see the top view of the EmbeddedMontiArc's IDE. The five added features here are as follows:

1. Open a new tab where you can play a normal game of Pacman
2. Generate the WebAssembly of the main component
3. Open a new tab in which the simulation of the component takes place
4. Generates the visualization of the main component and shows it in a new tab
5. Generates the reporting of all components and shows it in a new tab
6. Generates the reporting of all components with stream test results and shows it in a new tab
7. Run all tests in the repository and show their results

Figure 9: Main options for the Pacman project in the ide



8. Run a single test and show its result

The features needed to be implemented properly in different places in order to work along the logic of the ide. Each one calls a batch script which again runs the jar for the demanded task for the specific files. In addition, for feature 1 and 2 extra plugins were required which got implemented by the expert group Pacman and could be reused for Supermario.

4.2. Modeling (by Heithoff)

This chapter introduces the models of Pacman and Supermario respectively. The models should always follow certain rules defined in the EmbeddedMontiArc documentation (see [19]). The math implementation of all atomic components should be short and have a short runtime. This way not only the clarity of the code is enhanced but also the runtime of the components is fixed. C&C models should, at some point, be runnable on microchips and due to the fact that those models are designed for real-time systems the runtime has to be fix. To achieve this a lot of functionality can be extracted into subcomponents. In general, loops should be avoided and split up into subcomponents if possible. Because while loops are not ensured to terminate, those should never be used.

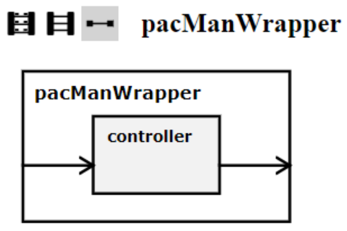
4.3. Pacman (by Heithoff)

In the following the model for the Pacman controller is presented. The goal is to collect as many biscuits and coins as possible and to avoid the ghosts. After introducing the interface which is used here two controllers for Pacman are shown. There is a simple controller which was used in the early stages of the IDE integration to test everything and then a more complex controller that can actually survive a few levels.

4.3.1. Interface

The project's main component is PacmanWrapper. The main task of the wrapper is to provide a shared interface. Listing 8 shows the input and output ports. As for the inputs, the ghosts' and the Pacman's position are given, the direction the ghosts are facing, information about the ghosts' vulnerability, as well as the current map. The only output port is the new direction the Pacman should walk.

Figure 10: Visualization of the Pacman wrapper



Listing 8: Interface of the Pacman Wrapper

```
ports
        in Z(0cm: 180cm) ghostX[4],
        in Z(0cm: 210cm) ghostY[4],
        in Z(0 : 1 : 3) ghostDirection[4],
        in B ghostEatable[4],
        in B ghostEaten[4],
        in Z(0cm: 180cm) pacManX,
        in Z(0cm: 210cm) pacManY,
        in B pacManEaten,
        in Z(0:oo) pacManLives,
        in Z(0:oo) pacManScore,
        in Z^{22,19} map,
        out Z(0 : 1 : 3) newPacmanDirection;
```

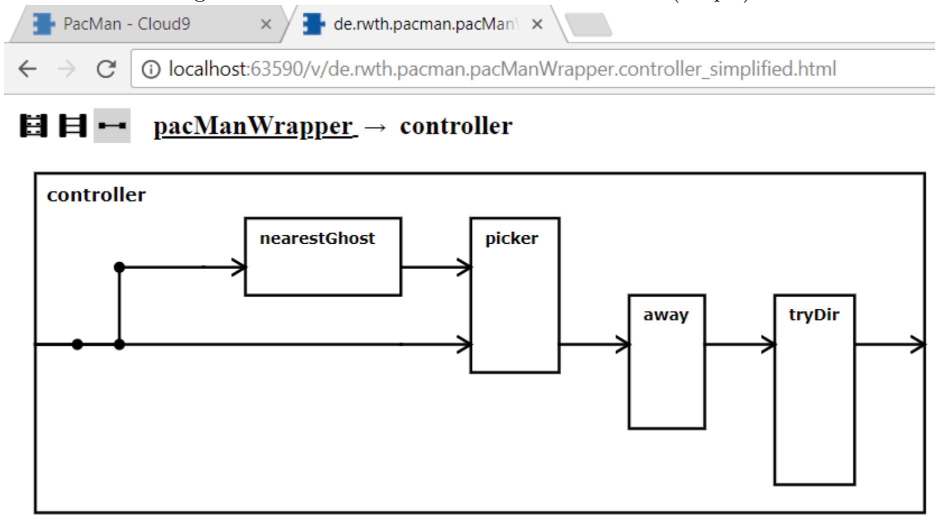
The wrapper also holds the current controller. This way the controller is easily exchangeable without changing any of the code needed for the ide. All input ports of the wrapper are connected to the corresponding ports of the controller and the output port of the controller is also connected to the output port of the wrapper.
To connect the web assembly of the main component with the Pacman emulator a new JavaScript file was created. Its main functionalities is to extract the needed informations out of the emulator, pass it to the web assembly, execute it and then give the output back to the emulator. In order to be able to extract needed information out of the emulator some modifications were needed. In its original state the emulator did not offer access to the current game object, thus the *Pacman* class was extended by these functions. Due to the fact Pacman is a playable game, its input is given as a key-press-event in JavaScript. So the output of the web assembly, which is a number from 0 to 3, is mapped to a corresponding key-press-event which then gets triggered. The emulator is running with 30 frames per second, which also leads to 30 iterations of the game per second. Because the emulator is running asynchronously the component is executed at a double of that rate in order to track every position change.

4.3.2. C&C modeling - Pacman (simple)

In figure 11 the design of a simple controller is shown. It has four subcomponents:

- nearestGhost: Is given the x - and y - position of every ghost and the x - and y - position of the Pacman. It then iterates over all ghosts and calculates the nearest ghost and gives back its index.
- picker: Is given all ghost informations as input as well as an index and gives back the ghost information of the ghost at this index.

Figure 11: Visualization of the Pacman controller (simple)


PacMan - Cloud9
de.rwth.pacman.pacMan
localhost:63590/v/de.rwth.pacman.pacManWrapper.controller_simplified.html
pacManWrapper → controller
controller
nearestGhost
picker
away
tryDir

- away: Is given one ghost's informations as well as Pacman's and calculates a new direction for the Pacman facing away from the ghost. The output is one of the four possible directions mapped from the numbers 0 to 3.
- tryDir: Gets as input the position of Pacman, the current map as well as a direction the Pacman should try to walk. If there is no wall blocking the way the initial direction is outputted. On the other hand, if there is a wall blocking the way it tries to walk orthogonally left or up. If it fails it will walk right or down respectively.

The controller connects the subcomponents in the shown order: It calculates the nearest ghost, passes its index to the picker which then again passes the corresponding ghost to the *away* component. This calculates the direction facing away from said ghost and the *tryDir* component then avoids running into walls. This leads to a controller that runs away from the ghosts with some success but it is only determined by the nearest ghost and has no other goals. Due to the fact that *tryDir* always tries to walk to the left (or top) first, this can lead to some stuttering as soon as the Pacman walked enough to the right that there is again space to the left.
This design is very simple and not very successful. It shows the concept of C&C modeling in its basics and is therefore listed here. The next controller is a lot more complex and can easily beat up to 10 levels.

4.3.3. C&C modeling - Pacman (complex)

The more complex Pacman controller is shown in figure 12. It has three main subcomponents:

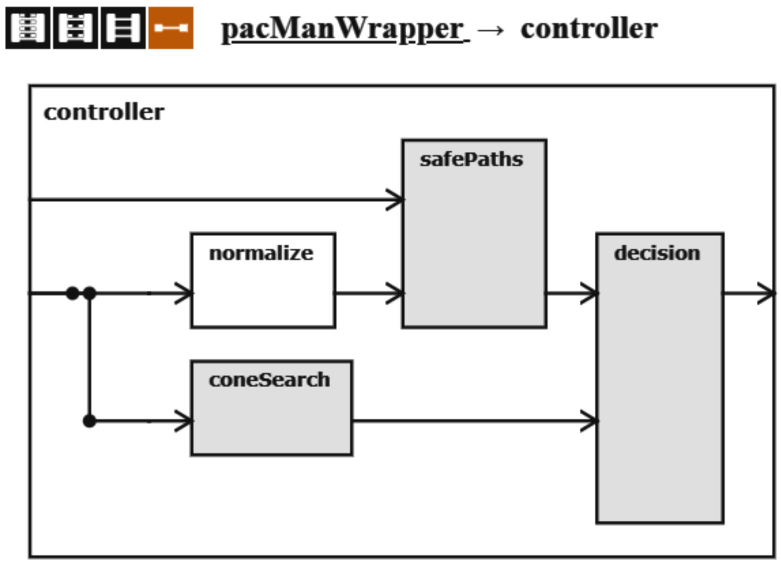


Figure 12: Visualization of the Pacman controller (complex)

- *safePaths*: This component is responsible for checking all the paths leading from Pacman into the labyrinth for safety. This is done by searching in each of the four possible directions until a wall or intersection is found.

- *coneSearch*: Searches in cones in each of the four directions for enemies and coins and gives back a score for each direction.

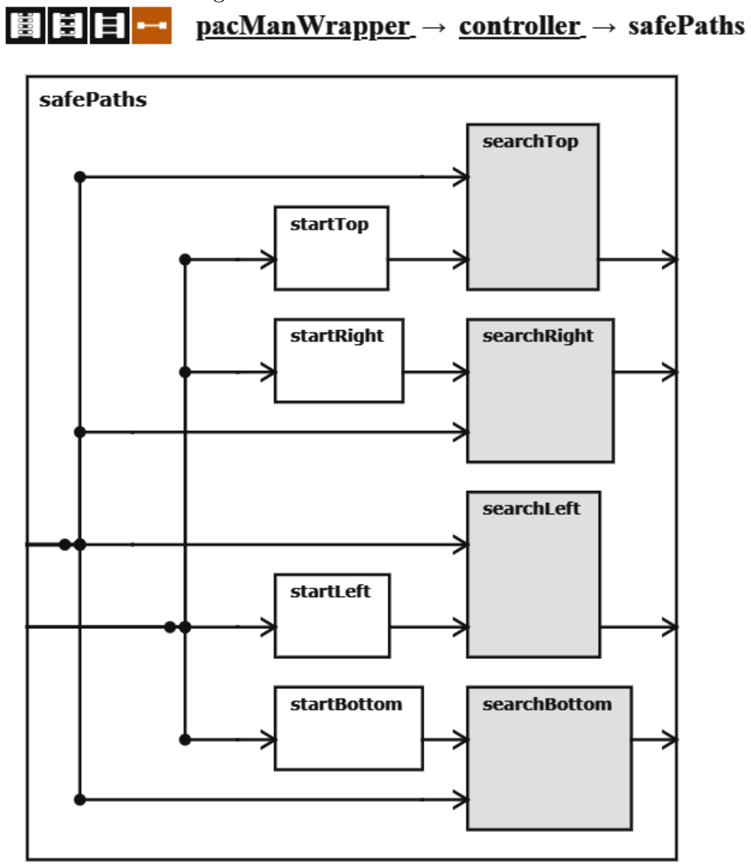
- *decision*: The decision component evaluates all data from the other two components. Based on those values it decides which direction the Pacman should go next.

The last component *normalize* not listed here is only responsible for increasing all position values from the ghosts and Pacman by 1 to fit the indexation from the math library. We will now go into detail for the three main component.

Safe Paths

In figure 13 the *safePaths* component is shown. It contains a subcomponent for each direction and some starting values. It gives back whether the four directions are safe or not. A direction is safe if there is a wall blocking it (no path) or there is no enemy on its path until the next intersection. This is calculated by "going" the path. This could be done with a single component looping through the path to the next intersection. Due to the fact that this would contradict the conditions on C&C components stated before it is split up into subcomponents. Each path in this labyrinth has a length of at most 10. So the task is divided into 10 components as one can partially see in fig. 14. Each of those checks whether the current position is safe and then calculate the

Figure 13: Visualization of Safe Paths



position to check for the next component. This way the runtime is fixed and the code is better parallelizable.

The task of one of the 10 subcomponents is again split up into 5 subtasks (see fig. 15):

- *reenterMap*: If the previous component calculated a position outside of the map (e.g. leaving the map on the right through the tunnel), reenter the map on the other side.
- *safeFinished*: The search is finished if it is marked as finished by a previous search component or a wall is found (only when there is no path).
- *safePosition*: Loops through the four ghosts and check whether their position matches the current position. If an unsafe tile is found, the search is marked as finished and not safe.
- *calcNewPosition*: Looks for free ways in the adjacent tiles. If there are more than two free tiles (no wall), an intersection is reached and the search can be marked finished. Otherwise this component gives back the next position which is different from the previous one.
- *control*: The control unit evaluates all data from the other components and gives back a corresponding new position and whether the search until now is safe or not.

Cone Search

The *ConeSearch* component searches through the map in cones (see figure 16). This way each direction can be given a value which increases when biscuits and coins are found and decreases when ghosts are found. The following weights are used in the most current version:

- biscuit: 50
- coin: 200
- enemy (facing towards Pacman): -10
- enemy (facing a different direction): -4
- enemy (eatable): 1000

The values shrink with the distance to Pacman. The biscuit/coin value shrink squared and the enemy value linear with the distance. This way near objectives are valued more and Pacman does not go for only far away biscuits/coins if there already are nearby ones. But if all biscuit/coin values are small the maximum gets increased by a fix amount, so Pacman goes for far away biscuits/coins if there are no around. In the end for each direction a value is returned by combining the biscuits/coins value and the enemy value.

In the visualization of the component (see fig. 17) one can see the different kind of subcomponents:

Figure 14: Visualization of one Search

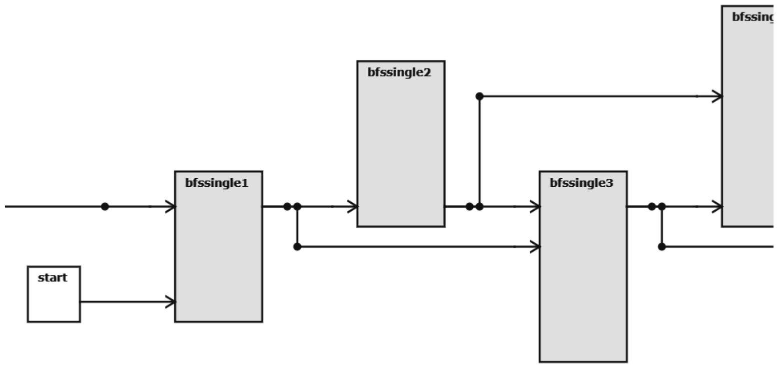


Figure 15: Visualization of one Single Search Component

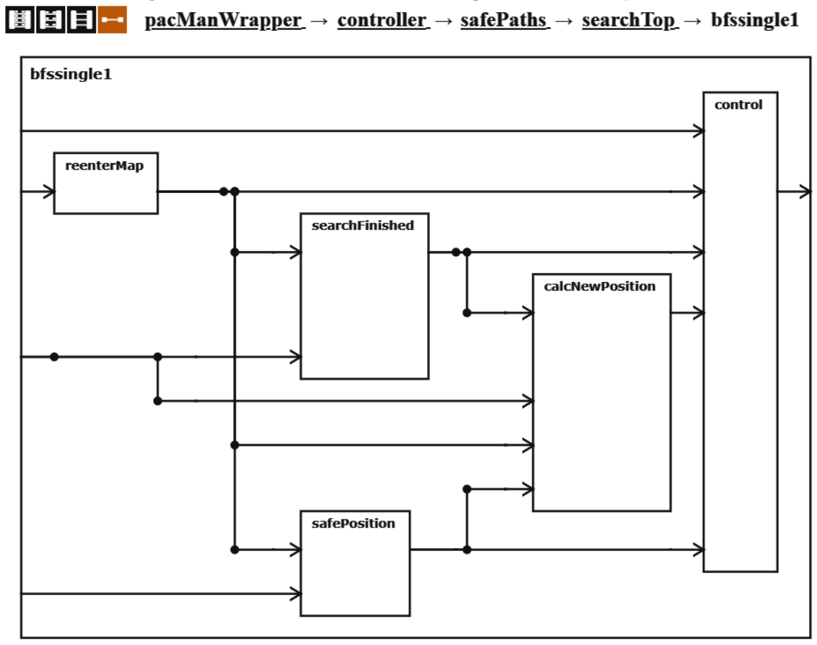


Figure 16: Visualization of Cone Search

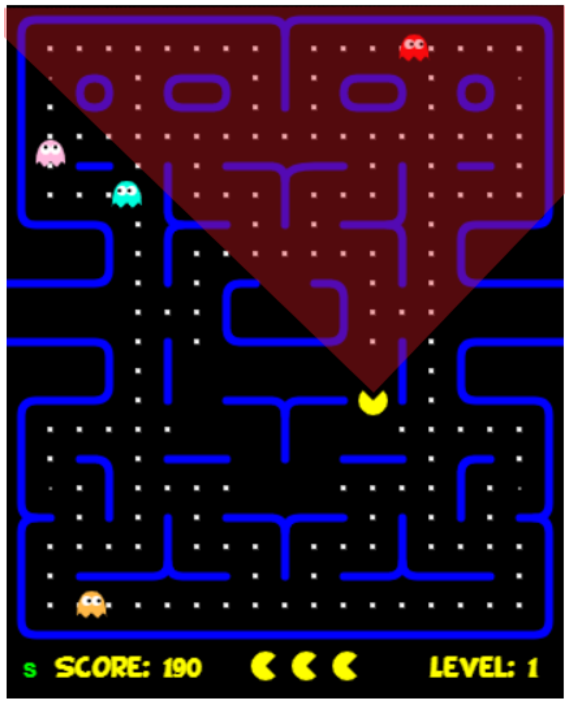
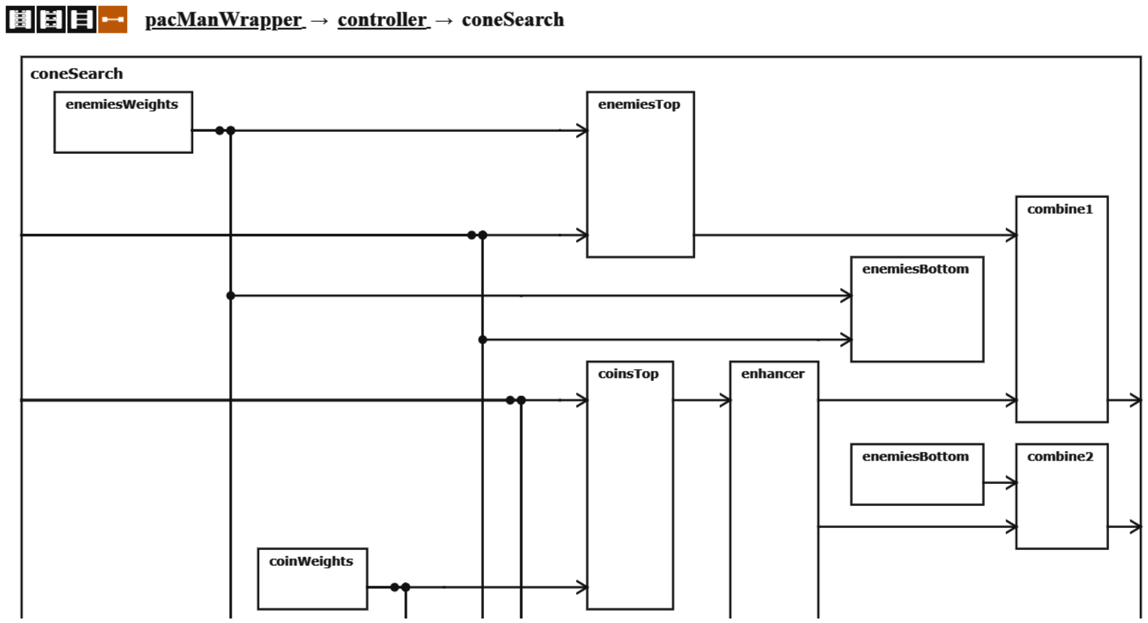


Figure 17: Visualization of Cone Search



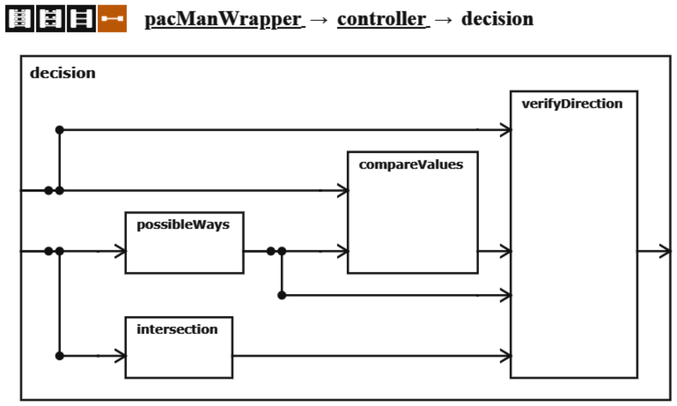
- *enemiesWeights* and *coinWeights*: some constants for weighting biscuits, coins and ghost values. This design allows easy adjustments.
- *enemies(Top)*: searches for enemies in the (top) cone and gives back its value.
- *coins(Top)*: searches for biscuits/coins in the (top) cone and gives back its value.
- *enhancer*: increases the maximum biscuits/coins value if it is small.
- *combine*: combines the values for biscuits/coins and enemies for a direction.

Decision

The *decision* component gets all data from *safePaths* and *coneSearch* and makes a final decision on where to go. Beside the maximum value for a direction and whether it is safe or not, the decision is based on a few additional information. E.g. the top direction has the maximum value from the cone searches but it is blocked by a wall or not safe. Then another direction has to be chosen. Here an orthogonal direction (left or right) is preferred to stay near to the desired one (top). In addition, to prevent stuttering a new path is only chosen if the current one is not safe anymore or an intersection is reached. In fig. 18 one can see the four subcomponents of *safePaths*:

- *intersection*: Gives back whether Pacman is located on a tile with more than 2 Paths leading from it.
- *possibleWays*: Gives back which directions are not blocked by a wall.
- *compareValues*: Calculates the safe direction with the maximum value. If this direction is blocked, a new direction has to be chosen.
- *verifyDirection*: Checks whether the chosen direction is opposing the previous one. This is only allowed if the previous direction is not safe anymore or an intersection is reached.

Figure 18: Visualization of Cone Search



4.4. Modeling - Supermario (by Philipp Haller)

This part discusses the model used to solve a level of the Supermario game. First a general introduction on model types is given. Thereafter, the different models are discussed step by step, beginning at the most abstract.

4.4.1. Model Types

In this context the following model types used were:

Watcher
The watcher model type takes a position as input and returns a boolean value which indicates if it is in a certain range.

Selector
The selector model type uses a raw array and an index as input and returns the corresponding array entry.

Strategy
A strategy model type can take different inputs and performs a action decision based on its inputs.

Controller
The controller model type combines the other defined model types to refine the inputs of the simulation and executes a strategy.

Filter
The filter model type is intended to perform filtering like debouncing and plausibility checks.

4.4.2. Models

The presented model visualizations are generated from the EmbeddedMontiArc Studio. Therein, a grey component indicates that the component uses additional subcomponents, whereas a white component marks atomic components. As stated before, the modeling was performed in such a way, that small components with few lines of code were preferred to bigger components.

The first and most abstract entity modeled was the Supermario wrapper which is closely related to the outputs and inputs of the simulator. Therefore it receives all necessary values as input with the aim to forward them to the actual controller and its corresponding sub-components. After computation the results of the controller are handed back into the wrapper, which forwards the data to the simulator. Figure 19 shows the graphical representation, while listing 9 shows the actual EMA interface definition.

The player figure's position, velocity and height were chosen as inputs, together with the positions of the next five enemies and obstacles. Furthermore, the position of the next hole in the ground, the position of the next five loot crates, the tick size (the time between model executions) and the information if the player is resting on a tile is given. The outputs consist of the direction the player shall go in combination with the action instructions jumping, crouching and shooting. The data type for most values is integer, indicated by a "Z" in the code. This is due to the circumstance that the simulator uses a number of pixels as a measure for distance. Only exception being the "tickSize" which can be fractions of a second.

The controller used (Figure 20) consists of five parts. There are sub-controllers tasked to cope with the evaluation of enemies and obstacles respectively, named enemyController and obstController. They return an advice to indicate if the player should jump or not. The genStrategy is an atomic component which is currently used to provide a general strategy like moving in another direction, jumping or crouching if the player is stuck.

The action advices of the controllers and the strategy are combined via a logical or-relation, as indicated by the "orR" block. Additionally, the jumpDecider filters the output of the combined value and forwards it, if the player can jump in that time frame. This is necessary to prohibit side-effects like the player only jumping once because the jump key remains pressed constantly and the simulator only accepts distinct jump activations, opposed to continuous jumping.

The enemy controller (Figure 21) handles the enemy position evaluation and assesses if an action has to be initiated. As the input data from the simulator is a array with five positions, it contains a enemy selector component which returns the corresponding x and y values from a given index. For purposes of overview and readability of the EMA code a component "enemyIndexes" was used to feed these indexes into the selectors.

The enemy component (Figure 22) is used to compute a velocity from the x and y positions by comparing the former positions with the current ones.

The enemy strategy (Figure 23) uses the distances and velocities from the

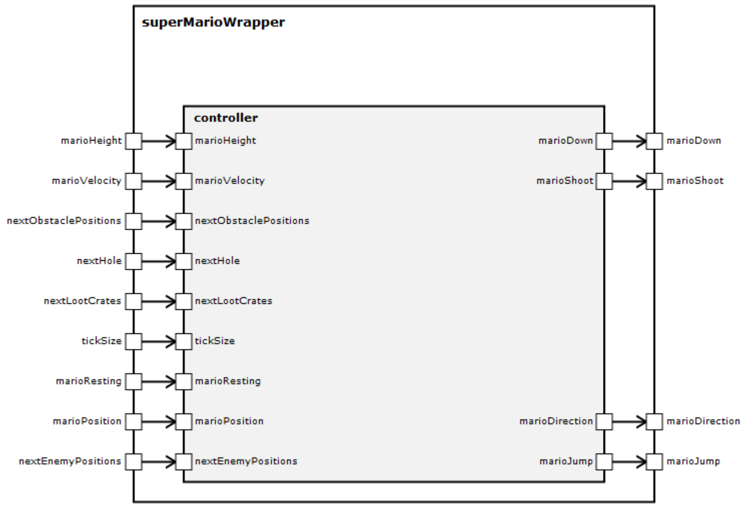


Figure 19: Visualisation of the Supermario wrapper model

Listing 9: Interface of the Supermario Wrapper

```
component SuperMarioWrapper {
    ports
        in Z^{1,2} marioPosition,
        in Z^{1,2} marioVelocity,
        in Z marioHeight,
        in Z^{5,2} nextEnemyPositions,
        in Z^{5,2} nextObstaclePositions,
        in Z nextHole,
        in Z^{5,2} nextLootCrates,
        in Q tickSize,
        in Z marioResting,
        out Z(−1 : 1 : 1) marioDirection,
        out Z marioJump,
        out Z marioDown,
        out Z marioShoot;
}
```

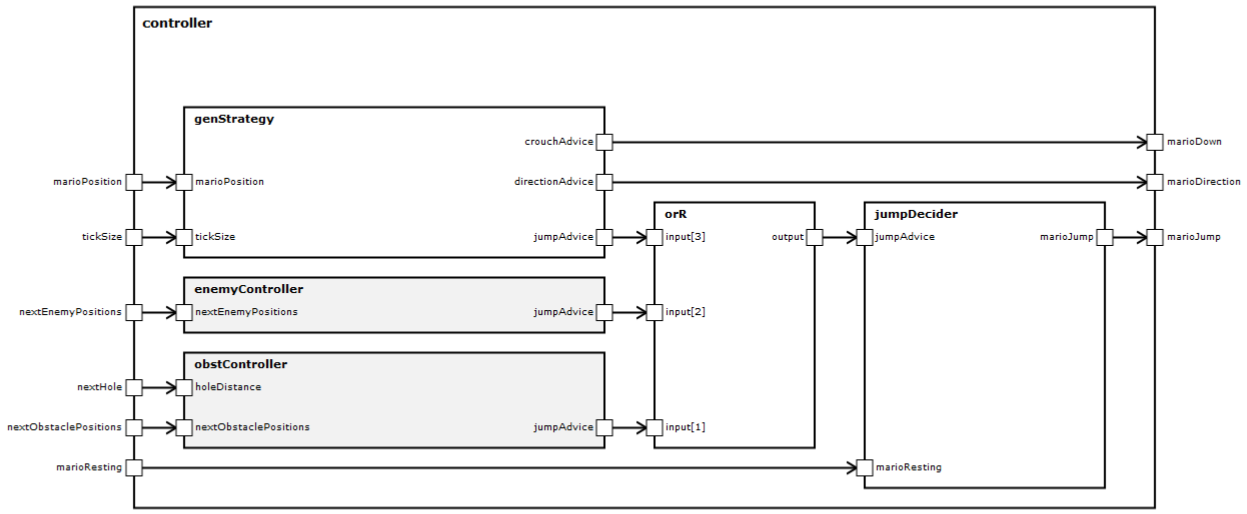


Figure 20: Visualization of the Supermario controller model

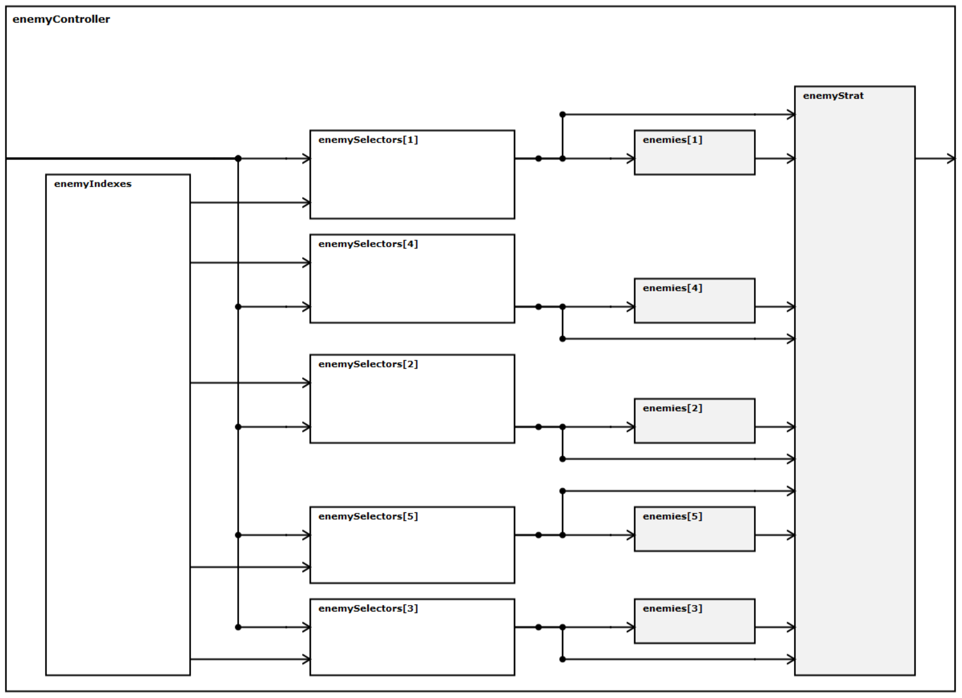


Figure 21: Visualization of the Supermario enemy controller model

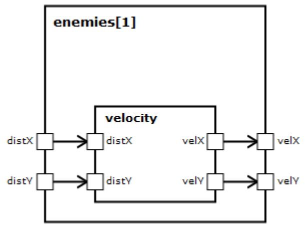


Figure 22: Visualization of the Supermario enemy model

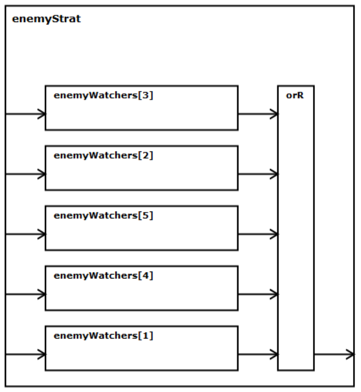


Figure 23: Visualization of the Supermario enemy strategy model

enemy components to watch them for their distance to the player and whether they can get dangerous. If an enemy comes too close and is on the player's plane, a jump advice is given. The jump advices are again combined via a logical or-relation and returned.

The obstacle controller is modeled very similar to the enemy controller, extracting positions from the raw input array and feeding them into a obstacle strategy. The main difference to the enemy controller is the presence of another input. This additional input is the distance to the next hole in the ground plane of the level. It is forwarded into the obstacle strategy (Figure 24) where a watcher component checks the player's proximity to the hole and computes a jump advice. All advices are again combined by a or relation.

4.4.3. Future Modeling

The models presented in this chapter were developed with modularity and extensibility in mind, such that in future work more complex strategies can be used to solve more levels and to lay more attention to the score. The presented model utilizes that the player always runs into the right direction, thus it can't solve levels which require the player to move backwards. A future model should be able to solve those situations too. This behavior could be modeled in the general strategy component or a "movement controller". Another issue could be, that currently all advices are combined via or relations. This can lead to side effects where the player jumps to early because of an enemy and drops into a hole he would have avoided without the enemy. To achieve a better model, the or relations could be swapped with a weighted decision making process.

5. Evaluation (by Malte Heithoff)

The two developers of Pacman and Supermario were interviewed in the manner mentioned in the introduction. In this section the results of this interviews are collected and summarized.

5.1. RQ1 - Is EmbeddedMontiArc suitable for other systems?

The tasks were both based on real-time problems which the EmbeddedMontiArc language is designed for. Both developers were able to model a controller which can beat a level in their specific domain. The code for the models is clearly and good readable. The generated Javascript code is fast enough to be executed every tick of the simulation (30 fps/ 60 fps). Based on the examples of Pacman and Supermario it is clear that real-time problems can be solved with EmbeddedMontiArc.

5.2. RQ2 - Is it possible to integrate other simulators in a recent amount of work?

To integrate the Pacman- and Supermario simulators two tasks had to be completed: integrate into the integrated development environment(IDE) and then link the simulator to the web assembly. The integration into the IDE was

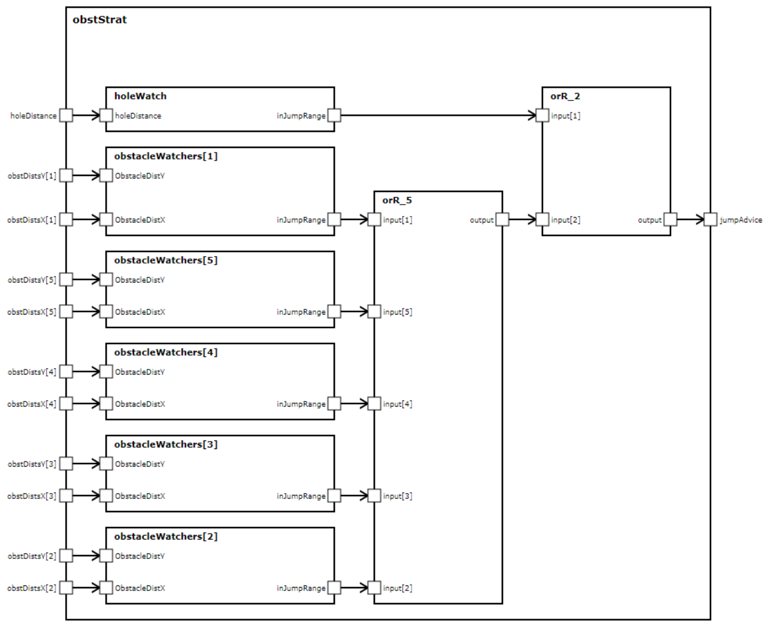


Figure 24: Visualization of the Supermario enemy strategy model

quite simple for both systems as soon as the instructions were handed out. But as there was no infrastructure for generating the web assembly before this led to some extra effort by installing emscripten and writing the needed scripts. The Simulator for Pacman was easily adjustable so that the extraction of the needed information (e.g., Pacman position) was done in short time. In contrast, the underlying structure of the Supermario project in use was way more complex and needed a lot more effort to understand it. Linking the web assembly with the simulator was done within little work as soon as the interface for extracting the data from the simulators and inputting the computed results was implemented. Just the data had to be transformed into the correct format and then the web assembly needed to be executed.

Therefore, the answer to this question is dependent on the complexity of the system and on whether the is a working interface for extracting data. Pacman was fairly easy to integrate but Supermario needed more time than calculated.

5.3. RQ3 - What kind of background knowledge is needed to model C&C in EmbeddedMontiArc?

One of the developers had some experience with EmbeddedMontiArc while the other had not. Both are computer science students and are therefore familiar with programming concepts and the modular programming that EmbeddedMontiArc requires. For the more experienced developer the concept of C&C was easy to understand and he could easily make use some of the tooling the language offers. The less experienced developer had a few problems in the beginning but after overcoming those he had no further problems with implementing what he was trying to. Both developers benefited from being familiar with programming languages so the math library was easy to understand.

Having experience with programming concepts is necessary to model C&C in EmbeddedMontiArc but specific knowledge about the EmbeddedMontiArc language is optional and can be obtained in a short time.

5.4. RQ4 - What features are good and what are not suited?

This section will be split up into the question about the tools around the language and the question about the features the language itself is offering.

5.4.1. Tools

The onlineIDE coming with the EmbeddedMontiArcStudio is powerful enough to help with the modeling process. But it is also missing a lot of tools a modern IDE is offering. The safe option was also one of the weak points of the IDE, only after running a plugin all the files are saved to the hard drive. Syntax checking was sufficient for the non-atomic components but missing for the atomic components. The other tools integrated into the IDE, such as generating a report with semantical checks of the models or generating a visualization could be utilized for error checking and planning the model. But most of the tools had a long runtime and need optimization.

5.4.2. Language Features

The option to import other components and to have a package hierarchy were used all the time and are well suited for the purpose of the language. Also connecting arrays of ports with a [:] is very convenient, but this option is not nested which made the code at some point larger than necessary. What was missing in this version of the code generator is the ability to use structs as a port type which led to unclear port interface for some components.

All in all, the features the IDE and the language were offering helped with the modeling process and are well suited for the language purpose.

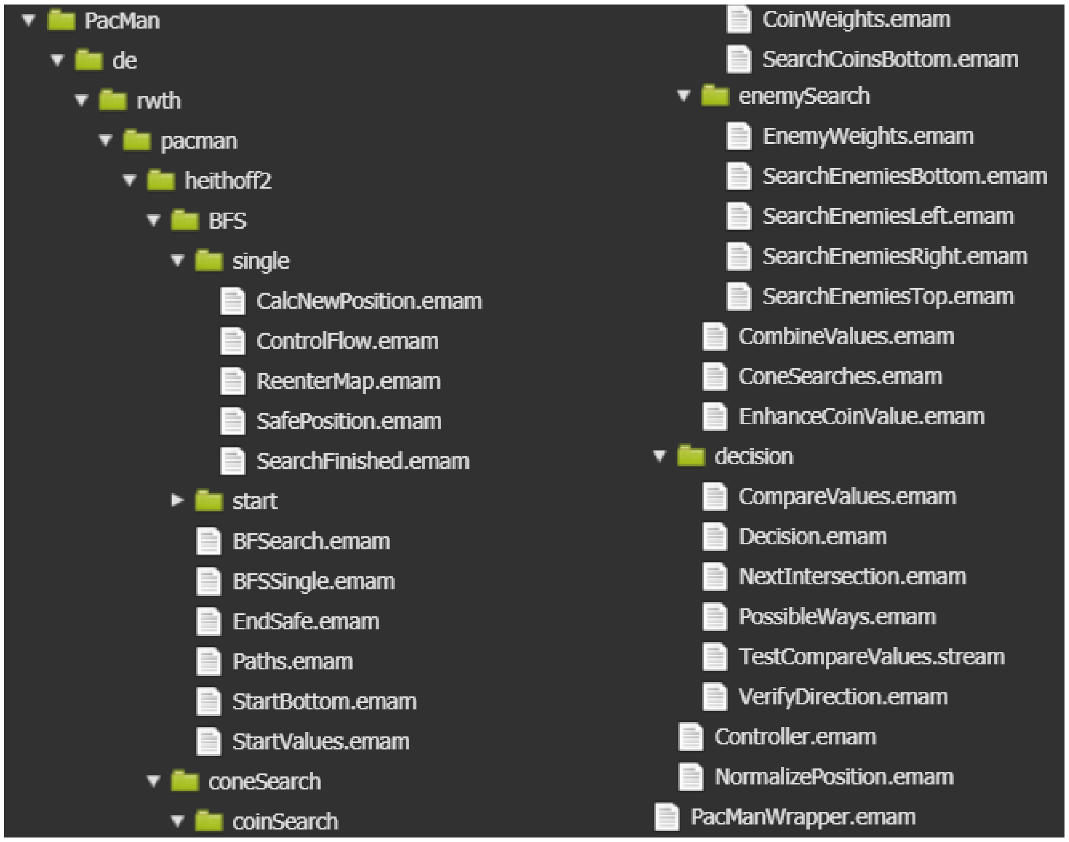
5.5. Other Problems

Although in theory there are no major problems with modeling the two groups had to fight some bugs in the code generation process. Most of those bugs are fixed by now, but at the time of modeling led to some considerable time losses. Due to the fact that the transformation from the C++ code to Javascript had a really long runtime, testing the code needed a lot of time as well.

6. Conclusion (by Haller and Heithoff)

In this paper, the question after the suitability of the DSL EmbeddedMontiArc for other systems different from the autopilot project is answered. For that reason the four research questions mentioned in the beginning were formulated. Those questioned are approached by assigning two groups the task to model a controller for the two real-time problems *Pacman* and *Supermario*. After successful modeling the development experience was condensed and presented. It showed that EmbeddedMontiArc as a language is suitable and intuitive, while the used integrated development environment and some bugs did cost a lot of time. Given a modern development environment is used, EmbeddedMontiArc has great potential towards reactive cyber-physical systems.

Appendix A. Pacman EmbeddedMontiArc Code



```
▼ PacMan
  ▼ de
    ▼ rwth
      ▼ pacman
        ▼ heithoff2
          ▼ BFS
            ▼ single
                CalcNewPosition.emam
                ControlFlow.emam
                ReenterMap.emam
                SafePosition.emam
                SearchFinished.emam
            ► start
              BFSearch.emam
              BFSSingle.emam
              EndSafe.emam
              Paths.emam
              StartBottom.emam
              StartValues.emam
          ▼ coneSearch
            ▼ coinSearch
                CoinWeights.emam
                SearchCoinsBottom.emam
            ▼ enemySearch
                EnemyWeights.emam
                SearchEnemiesBottom.emam
                SearchEnemiesLeft.emam
                SearchEnemiesRight.emam
                SearchEnemiesTop.emam
              CombineValues.emam
              ConeSearches.emam
              EnhanceCoinValue.emam
          ▼ decision
              CompareValues.emam
              Decision.emam
              NextIntersection.emam
              PossibleWays.emam
              TestCompareValues.stream
              VerifyDirection.emam
            Controller.emam
            NormalizePosition.emam
          PacManWrapper.emam
```

Figure A.25: Pacman package outline

Appendix B. Pacman Stream Test Code

Listing 10: PacmanWrapper

```
package de.rwth.pacman;
import de.rwth.pacman.heithoff2.Controller;
import de.rwth.pacman.structs.Ghost;

// UP = 0
// DOWN = 1
// LEFT = 2
// RIGHT = 3

component PacManWrapper {
    ports
        in Z(−1m: 19m) ghostX[4],
        in Z(0m: 21m) ghostY[4],
        in Z(0 : 1 : 3) ghostDirection[4],
        in B ghostEatable[4],
        in B ghostEaten[4],
        in Z(−1m: 19m) pacManX,
        in Z(0m: 21m) pacManY,
        in B pacManEaten,
        in Z(0:oo) pacManLives,
        in Z(0:oo) pacManScore,
        in Zˆ{22,19} map,

        out Z(0 : 1 : 3) newPacManDirection;

    //Replace this with your own custom controller
    instance Controller controller;

    connect ghostX[:] → controller.ghostX[:];
    connect ghostY[:] → controller.ghostY[:];
    connect ghostDirection[:] → controller.ghostDirection[:];
    connect ghostEatable[:] → controller.ghostEatable[:];
    connect ghostEaten[:] → controller.ghostEaten[:];
    connect pacManX → controller.pacManX;
    connect pacManY → controller.pacManY;
    connect pacManEaten → controller.pacManEaten;
    connect pacManLives → controller.pacManLives;
    connect pacManScore → controller.pacManScore;
    connect map → controller.map;

    connect controller.newPacManDirection → newPacManDirection;
}
```

Listing 11: Controller

```
package de.rwth.pacman.heithoff2;

import de.rwth.pacman.heithoff2.BFS.Paths;
import de.rwth.pacman.heithoff2.decision.Decision;
import de.rwth.pacman.heithoff2.coneSearch.ConeSearches;

component Controller {
    ports
      in Q(−1m: 19m) ghostX[4],
      in Q(0m: 22m) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],
      in B ghostEaten[4],
      in Q(−1m: 19m) pacManX,
      in Q(0m: 22m) pacManY,
      in B pacManEaten,
      in Z(0:oo) pacManLives,
      in Z(0:oo) pacManScore,
      in Zˆ{22,19} map,
      out Z(0 : 1 : 3) newPacManDirection;

    instance Paths safePaths; // gives back whether certain paths are safe
    instance Decision decision; // main strategy
    instance ConeSearches coneSearch; // searches for coins and enemies
    instance NormalizePosition normalize;

    connect ghostX[:] → normalize.ghostX[:], coneSearch.ghostX[:];
    connect ghostY[:] → normalize.ghostY[:], coneSearch.ghostY[:];
    connect ghostDirection[:] → safePaths.ghostDirection[:], coneSearch.
         ghostDirection[:];
    connect ghostEatable[:] → safePaths.ghostEatable[:], coneSearch.
         ghostEatable[:];
    connect pacManX → normalize.pacManX, decision.pacManX,
         coneSearch.currentX;
    connect pacManY → normalize.pacManY, decision.pacManY,
         coneSearch.currentY;
    connect map → safePaths.map, decision.map, coneSearch.map;
    connect normalize.newPacManX → safePaths.pacManX;
    connect normalize.newPacManY → safePaths.pacManY;
    connect normalize.newGhostX[:] → safePaths.ghostX[:];
    connect normalize.newGhostY[:] → safePaths.ghostY[:];

    connect safePaths.topSafe → decision.topSafe;
    connect safePaths.bottomSafe → decision.bottomSafe;
    connect safePaths.leftSafe → decision.leftSafe;
    connect safePaths.rightSafe → decision.rightSafe;

    connect coneSearch.topValue → decision.topValue;
    connect coneSearch.bottomValue → decision.bottomValue;
    connect coneSearch.leftValue → decision.leftValue;
    connect coneSearch.rightValue → decision.rightValue;

    connect decision.newPacManDirection → newPacManDirection;
}
```

Listing 12: NormalizePosition

```
package de.rwth.pacman.heithoff2;

component NormalizePosition {
    ports
      in Q(−1m: 19m) pacManX,
      in Q(0m: 22m) pacManY,
      in Q(−1m: 19m) ghostX[4],
      in Q(0m: 22m) ghostY[4],
      out Q(−1m: 19m) newPacManX,
      out Q(0m: 22m) newPacManY,
      out Q(−1m: 19m) newGhostX[4],
      out Q(0m: 22m) newGhostY[4];

    implementation Math {
        newPacManX = pacManX + 1;
        newPacManY = pacManY + 1;
        for i = 1:4
            newGhostX(i) = ghostX(i) + 1;
            newGhostY(i) = ghostY(i) + 1;
        end
    }
}
```

Listing 13: BFS.BFSearch

```
package de.rwth.pacman.heithoff2.BFS;

import de.rwth.pacman.heithoff2.BFS.start.StartValues;

// search along the current path whether there are ghosts facing to pacman
// bfssingle1 is given the start values which then calculates the next
     coordinates
// for bffsingle2
// the path ends when an intersection is reached
// then check again whether the surrounding tiles are safe
component BFSearch{
    ports
        in Q(0m: 20m) ghostX[4],
        in Q(1m: 23m) ghostY[4],
        in Q(0m: 20m) pacManX,
        in Q(1m: 23m) pacManY,
        in Z(0 : 1 : 3) ghostDirection[4],
        in B ghostEatable[4],
        in Z^{22,19} map,
        in Q(0m: 20m) startX,
        in Q(1m: 23m) startY,
        in Z startDirection,

        out B safe;

    instance StartValues start;
    instance BFSSingle bfssingle1;
    instance BFSSingle bfssingle2;
    instance BFSSingle bfssingle3;
    instance BFSSingle bfssingle4;
    instance BFSSingle bfssingle5;
    instance BFSSingle bfssingle6;
    instance BFSSingle bfssingle7;
    instance BFSSingle bfssingle8;
    instance BFSSingle bfssingle9;
    instance BFSSingle bfssingl9;
    instance EndSafe endSafe;

    connect pacManX → bfssingle1.oldX;
    connect pacManY → bfssingle1.oldY;

    connect startX → bfssingle1.currentX, bfssingle2.oldX;
    connect startY → bfssingle1.currentY, bfssingle2.oldY;
    connect start.startSafe → bfssingle1.oldSafe;
    connect start.startSafeFound → bfssingle1.oldSafeFound;
    connect startDirection → bfssingle1.oldDirection;

    connect bfssingle1.newX → bfssingle2.currentX, bfssingle3.oldX;
    connect bfssingle1.newY → bfssingle2.currentY, bfssingle3.oldY;
    connect bfssingle1.safe → bfssingle2.oldSafe;
    connect bfssingle1.safeFound → bfssingle2.oldSafeFound;
    connect bfssingle1.newDirection → bfssingle2.oldDirection;

    connect bfssingle2.newX → bfssingle3.currentX, bfssingle4.oldX;
    connect bfssingle2.newY → bfssingle3.currentY, bfssingle4.oldY;
```

Listing 14: BFS.BFSSingle

```
package de.rwth.pacman.heithoff2.BFS;
import de.rwth.pacman.heithoff2.BFS.single.∗;
// check whether the current tile is safe and then calculate the next tile position
component BFSSingle {
    ports
        in Q(0m: 20m) ghostX[4],
        in Q(1m: 23m) ghostY[4],
        in Z(0 : 1 : 3) ghostDirection[4],
        in B ghostEatable[4],
        in Zˆ{22,19} map,
        in Q(0m: 20m) currentX,
        in Q(1m: 23m) currentY,
        in Q(0m: 20m) oldX,
        in Q(1m: 23m) oldY,
        in B oldSafe,
        in B oldSafeFound,
        in Z oldDirection,
        out Q(0m: 20m) newX,
        out Q(1m: 23m) newY,
        out B safeFound,
        out B safe,
        out Z newDirection;

    instance ControlFlow control;
    instance ReenterMap reenterMap;
    instance SearchFinished searchFinished;
    instance SafePosition safePosition;
    instance CalcNewPosition calcNewPosition;
    connect currentX → reenterMap.currentX;
    connect currentY → reenterMap.currentY;
    connect oldX → reenterMap.oldX;
    connect oldY → reenterMap.oldY;
    connect reenterMap.newCurrentX → searchFinished.currentX, safePosition.currentX,
          calcNewPosition.currentX, control.currentX;
    connect reenterMap.newCurrentY → searchFinished.currentY, safePosition.currentY,
          calcNewPosition.currentY, control.currentY;
    connect map → searchFinished.map, calcNewPosition.map;
    connect oldSafe → searchFinished.oldSafe;
    connect oldSafeFound → searchFinished.oldSafeFound;
    connect ghostX[:] → safePosition.ghostX[:];
    connect ghostY[:] → safePosition.ghostY[:];
    connect ghostDirection[:] → safePosition.ghostDirection[:];
    connect ghostEatable[:] → safePosition.ghostEatable[:];
    connect oldDirection → safePosition.oldDirection, control.oldDirection;
    connect reenterMap.newOldX → calcNewPosition.oldX;
    connect reenterMap.newOldY → calcNewPosition.oldY;
    connect searchFinished.finished → calcNewPosition.searchFinished;
    connect safePosition.safe → calcNewPosition.positionIsSafe;
    connect searchFinished.finished → control.searchFinished;
    connect searchFinished.safe → control.safeFromSearchFinished;
    connect searchFinished.safeFound → control.safeFoundFromSearchFinished;
    connect safePosition.safe → control.positionIsSafe;
    connect calcNewPosition.safeFound → control.safeFoundFromNewPosition;
    connect calcNewPosition.newX → control.newXFromNewPosition;
    connect calcNewPosition.newY → control.newYFromNewPosition;
    connect calcNewPosition.newDirection → control.newDirectionFromNewPosition;
    connect control.newX → newX;
    connect control.newY → newY;
    connect control.safeFound → safeFound;
    connect control.safe → safe;
    connect control.newDirection → newDirection;
}
```

Listing 15: BFS.EndSafe

```
package de.rwth.pacman.heithoff2.BFS;

// check whether the surrounding tiles are safe
component EndSafe {
    ports
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in Q(0m: 20m) ghostX[4],
      in Q(1m: 23m) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],
      in B oldSafe,
      in B oldSafeFound,
      in Z oldDirection,

      out B safe;

    implementation Math {
        Z^{1,4} xOffSet = [0,0,−1,1];
        Z^{1,4} yOffSet = [−1,1,0,0];

        safe = 1;
        if oldSafe
            for i = 1:4
                if (ghostEatable(i) == 0)
                    Z xG = round(ghostX(i));
                    Z yG = round(ghostY(i));
                    Z xC = currentX;
                    Z yC = currentY;
                    if (xG == xC) && (yG == yC)
                        safe = 0;
                    end
                    for j = 0:3
                        xC = currentX + xOffSet(0,j);
                        yC = currentY + yOffSet(0,j);
                        if (xG == xC) && (yG == yC) && (ghostEatable(i
                             ) == 0) && (ghostDirection(i) != j)
                            safe = 0;
                        end
                    end
                end
            end
        else
            safe = 0;
        end
    }
}
```

Listing 16: BFS.Paths

```
package de.rwth.pacman.heithoff2.BFS;
import de.rwth.pacman.heithoff2.BFS.start.∗;

// check whether the four directions are safe to go
// a directions is not safe to go if there is a ghost on its path
component Paths {
    ports
      in Q(0m: 20m) ghostX[4],
      in Q(1m: 23m) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],
      in Q(0m: 20m) pacManX,
      in Q(1m: 23m) pacManY,
      in Zˆ{22,19} map,

      out B topSafe,
      out B bottomSafe,
      out B leftSafe,
      out B rightSafe;

    instance BFSearch searchLeft;
    instance BFSearch searchRight;
    instance BFSearch searchTop;
    instance BFSearch searchBottom;
    instance StartLeft startLeft;
    instance StartRight startRight;
    instance StartTop startTop;
    instance StartBottom startBottom;

    connect pacManX → startLeft.pacManX, startRight.pacManX, startTop.pacManX,
         startBottom.pacManX;
    connect pacManY → startLeft.pacManY, startRight.pacManY, startTop.pacManY,
         startBottom.pacManY;

    connect ghostX[:] → searchLeft.ghostX[:], searchRight.ghostX[:], searchTop.ghostX[:],
         searchBottom.ghostX[:];
    connect ghostY[:] → searchLeft.ghostY[:], searchRight.ghostY[:], searchTop.ghostY[:],
         searchBottom.ghostY[:];
    connect ghostDirection[:] → searchLeft.ghostDirection[:], searchRight.ghostDirection[:],
          searchTop.ghostDirection[:], searchBottom.ghostDirection[:];
    connect ghostEatable[:] → searchLeft.ghostEatable[:], searchRight.ghostEatable[:],
         searchTop.ghostEatable[:], searchBottom.ghostEatable[:];
    connect map → searchLeft.map, searchRight.map, searchTop.map, searchBottom.map;

    connect pacManX → searchLeft.pacManX, searchRight.pacManX, searchTop.pacManX
         , searchBottom.pacManX;
    connect pacManY → searchLeft.pacManY, searchRight.pacManY, searchTop.pacManY
         , searchBottom.pacManY;
    connect startLeft.startX → searchLeft.startX;
    connect startLeft.startY → searchLeft.startY;
    connect startLeft.startD → searchLeft.startDirection;
    connect startRight.startX → searchRight.startX;
    connect startRight.startY → searchRight.startY;
    connect startRight.startD → searchRight.startDirection;
    connect startTop.startX → searchTop.startX;
    connect startTop.startY → searchTop.startY;
    connect startTop.startD → searchTop.startDirection;
    connect startBottom.startX → searchBottom.startX;
    connect startBottom.startY → searchBottom.startY;
    connect startBottom.startD → searchBottom.startDirection;

    connect searchLeft.safe → leftSafe;
    connect searchRight.safe → rightSafe;
    connect searchTop.safe → topSafe;
    connect searchBottom.safe → bottomSafe;
}
```

Listing 17: BFS.StartBottom

```
package de.rwth.pacman.heithoff2.BFS;

component StartBottom{
    ports
        in Z(−1m: 19m) pacManX,
        in Z(0m: 22m) pacManY,
        out Z(−1m: 19m) startX,
        out Z(0m: 22m) startY,
        out Z startD;

    implementation Math {
        startX = pacManX;
        startY = round(pacManY + 0.51);
        startD = 1;
    }
}
```

Listing 18: BFS.StartValues

```
package de.rwth.pacman.heithoff2.BFS;

component StartValues{
    ports
        out B startSafe,
        out B startSafeFound,
        out Z startValue;

    implementation Math {
        startSafe = 1;
        startSafeFound = 0;
    }
}
```

Listing 19: BFS.single.CalcNewPosition

```
package de.rwth.pacman.heithoff2.BFS.single;

component CalcNewPosition {
    ports
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in Q(0m: 20m) oldX,
      in Q(1m: 23m) oldY,
      in Z^{22,19} map,
      in B searchFinished,
      in B positionIsSafe,
      out Q(0m: 20m) newX,
      out Q(1m: 23m) newY,
      out Z newDirection,
      out B safeFound;

    implementation Math {
        newX = currentX;
        newY = currentY;
        newDirection = 0;
        safeFound = 0;
        if (searchFinished == 0) && (positionIsSafe == 1)
            // check for intersection or calculate the next tile
            Z^{1,4} xOffSet = [0,0,−1,1];
            Z^{1,4} yOffSet = [−1,1,0,0];
            safeFound = 0;
            Z newPathsFound = 0;
            for i = 0:3
                Z indexY = 0;
                Z indexX = i;
                Z xOff = xOffSet(indexY, indexX);
                Z yOff = yOffSet(indexY, indexX);
                Q xT = currentX + xOff;
                Q yT = currentY + yOff;

                if (abs(xT − oldX) >= 1) || (abs(yT − oldY) >= 1)
                    Z nextTile = map(yT, xT);
                    if (nextTile != 0) && (nextTile != 3) // a non−blocking
                         tile was found
                        newPathsFound = newPathsFound + 1;
                        newX = xT;
                        newY = yT;
                        newDirection = i;
                        if newPathsFound > 1
                            safeFound = 1;
                            newX = currentX;
                            newY = currentY;
                        end
                    end
                end
            end
        end
    }
}
```

Listing 20: BFS.single.ControlFlow

```
package de.rwth.pacman.heithoff2.BFS.single;

component ControlFlow {
    ports
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in B searchFinished,
      in B safeFromSearchFinished,
      in B safeFoundFromSearchFinished,
      in Z oldDirection,
      in B positionIsSafe,
      in B safeFoundFromNewPosition,
      in Q(0m: 20m) newXFromNewPosition,
      in Q(1m: 23m) newYFromNewPosition,
      in Z newDirectionFromNewPosition,

      out Q(0m: 20m) newX,
      out Q(1m: 23m) newY,
      out B safeFound,
      out B safe,
      out Z newDirection;

    implementation Math {
        newDirection = oldDirection;
        newX = currentX;
        newY = currentY;

        if searchFinished == 1
            safe = safeFromSearchFinished;
            safeFound = safeFoundFromSearchFinished;
        elseif positionIsSafe == 0
            safe = 0;
            safeFound = 0;
        else
            safe = 1;
            safeFound = safeFoundFromNewPosition;
            newX = newXFromNewPosition;
            newY = newYFromNewPosition;
            newDirection = newDirectionFromNewPosition;
        end
    }
}
```

Listing 21: BFS.single.ReenterMap

```
package de.rwth.pacman.heithoff2.BFS.single;

component ReenterMap {
    ports
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in Q(0m: 20m) oldX,
      in Q(1m: 23m) oldY,

      out Q(0m: 20m) newCurrentX,
      out Q(1m: 23m) newCurrentY,
      out Q(0m: 20m) newOldX,
      out Q(1m: 23m) newOldY;

    implementation Math {
        newCurrentX = currentX;
        newCurrentY = currentY;
        newOldX = oldX;
        newOldY = oldY;
        if currentX < 2
            newCurrentX = 18;
            newOldX = 19;
        elseif currentX > 18
            newCurrentX = 2;
            newOldX = 1;
        end
    }
}
```

Listing 22: BFS.single.SafePosition

```
package de.rwth.pacman.heithoff2.BFS.single;

component SafePosition {
    ports
      in Q(0m: 20m) ghostX[4],
      in Q(1m: 23m) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in Z oldDirection,

      out B safe;

    implementation Math {
        safe = 1;

        // check whether the current tile is safe
        for i = 1:4
            Z xG = round(ghostX(i));
            Z yG = round(ghostY(i));
            if (abs(xG − currentX) < 1) && (abs(yG − currentY) < 1) && (
                ghostEatable(i) == 0) && (ghostDirection(i) != oldDirection
                )
                safe = 0;
            end
        end
    }
}
```

Listing 23: BFS.single.SearchFinished

```
package de.rwth.pacman.heithoff2.BFS.single;

component SearchFinished {
    ports
      in Zˆ{22,19} map,
      in Q(0m: 20m) currentX,
      in Q(1m: 23m) currentY,
      in B oldSafe,
      in B oldSafeFound,

      out B safeFound,
      out B safe,
      out B finished;

    implementation Math {
        safeFound = oldSafeFound;
        safe = oldSafe;
        finished = 0;
        Z currentTile = 0;//map(currentY, currentX);
        if (currentY < 23) && (currentY > 0) && (currentX < 20) && (
             currentX > 0)
            currentTile = map(currentY, currentX);
        end
        if (currentTile == 0) || (currentTile == 3) // begin within a
             wall−tile, nothing to check
            safeFound = 1;
            safe = 1;
            finished = 1;
        elseif (oldSafeFound == 1) || (oldSafe == 0) // already at an
             intersection or a ghost was found
            finished = 1;
        end
    }
}
```

Listing 24: coneSearch.CombineValues

```
package de.rwth.pacman.heithoff2.coneSearch;

// a simple add

component CombineValues {
    ports
      in Z valueCoins,
      in Z valueEnemies,

      out Z value;

    implementation Math {
        // some weightning here

        value = valueCoins + valueEnemies;
    }
}
```

Listing 25: coneSearch.ConeSearches

```
package de.rwth.pacman.heithoff2.coneSearch;

import de.rwth.pacman.heithoff2.coneSearch.coinSearch.∗;
import de.rwth.pacman.heithoff2.coneSearch.enemySearch.∗;

// Search in cones to all four directions for coins/buiscuits and enemies

component ConeSearches {
    ports
      in Z^{22,19} map,
      in Q(−1m: 19m) currentX,
      in Q(0m: 22m) currentY,
      in Q(−1m: 19m) ghostX[4],
      in Q(0m: 22m) ghostY[4],
      in Z(0 : 1 : 3) ghostDirection[4],
      in B ghostEatable[4],

      out Z topValue,
      out Z bottomValue,
      out Z leftValue,
      out Z rightValue;

    instance SearchCoinsTop coinsTop;
    instance SearchCoinsBottom coinsBottom;
    instance SearchCoinsLeft coinsLeft;
    instance SearchCoinsRight coinsRight;
    instance CoinWeights coinWeights;

    instance SearchEnemiesTop enemiesTop;
    instance SearchEnemiesBottom enemiesBottom;
    instance SearchEnemiesLeft enemiesLeft;
    instance SearchEnemiesRight enemiesRight;
    instance EnemyWeights enemiesWeights;

    instance EnhanceCoinValue enhancer;

    instance CombineValues combine1;
    instance CombineValues combine2;
    instance CombineValues combine3;
    instance CombineValues combine4;


    connect map → coinsTop.map, coinsBottom.map, coinsLeft.map,
        coinsRight.map;
    connect ghostX[:] → enemiesTop.ghostX[:], enemiesBottom.ghostX[:],
        enemiesLeft.ghostX[:], enemiesRight.ghostX[:];
    connect ghostY[:] → enemiesTop.ghostY[:], enemiesBottom.ghostY[:],
        enemiesLeft.ghostY[:], enemiesRight.ghostY[:];
    connect currentX → coinsTop.currentX, enemiesTop.currentX,
        coinsBottom.currentX, enemiesBottom.currentX, coinsLeft.currentX,
        enemiesLeft.currentX, coinsRight.currentX, enemiesRight.currentX;
    connect currentY → coinsTop.currentY, enemiesTop.currentY,
        coinsBottom.currentY, enemiesBottom.currentY, coinsLeft.currentY,
        enemiesLeft.currentY, coinsRight.currentY, enemiesRight.currentY;
    connect ghostEatable[:] → enemiesTop.ghostEatable[:], enemiesBottom.
```

Listing 26: coneSearch.EnhanceCoinValue

```
package de.rwth.pacman.heithoff2.coneSearch;

// enhance the buiscuit and coin value if they are too far away
// this way pacman finds distant buiscuits and coins as well

component EnhanceCoinValue {
    ports
      in Z valueIn[4],
      out Z valueOut[4];

    implementation Math {
        Z max = −1;
        Z index = −1;

        for i = 1:4
            if max < valueIn(i)
                max = valueIn(i);
                index = i;
            end
            valueOut(i) = valueIn(i);
        end
        if max < 10
            valueOut(index) = 100;
        end
    }
}
```

Listing 27: coneSearch.coinSearch.CoinWeights

```
package de.rwth.pacman.heithoff2.coneSearch.coinSearch;

component CoinWeights {
    ports
      out Z buiscuitWeight,
      out Z coinWeight;

    implementation Math {
        buiscuitWeight = 50;
        coinWeight = 200;
    }
}
```

Listing 28: coneSearch.coinSearch.SearchCoinsBottom

```
package de.rwth.pacman.heithoff2.coneSearch.coinSearch;

component SearchCoinsBottom {
    ports
      in Zˆ{22,19} map,
      in Q(−1m: 19m) currentX,
      in Q(0m: 22m) currentY,
      in Z buiscuitWeight,
      in Z coinWeight,

      out Z value;

    implementation Math {
        value = 0;
        for i = 1:21
            Z indexY = round(currentY) + i + 1;
            if indexY < 22
                for j = (−i):i
                    Z indexX = round(currentX) + j + 1;
                    if (indexX > 0) && (indexX < 19)
                        Z nextTile = map(indexY, indexX);
                        if (nextTile == 1) || (nextTile == 4)
                            Z multBy = 1;
                            if nextTile == 1
                                multBy = buiscuitWeight;
                            elseif nextTile == 4
                                multBy = coinWeight;
                            end
                            Q dist = sqrt(i∗i + j∗j);
                            value = value + multBy/(dist∗dist);
                        end
                    end
                end
            end
        end
    }
}
```

Listing 29: coneSearch.enemySearch.EnemyWeights

```
package de.rwth.pacman.heithoff2.coneSearch.enemySearch;

component EnemyWeights {
    ports
      out Z normal,
      out Z towardsPacMan,
      out Z eatable;

    implementation Math {
        normal = −4;
        towardsPacMan = −10;
        eatable = 5000;
    }
}
```

Listing 30: coneSearch.enemySearch.SearchEnemiesBottom

```
package de.rwth.pacman.heithoff2.coneSearch.enemySearch;

component SearchEnemiesBottom {
    ports
      in Q(−1m: 19m) currentX,
      in Q(0m: 22m) currentY,
      in Q(−1m: 19m) ghostX[4],
      in Q(0m: 22m) ghostY[4],
      in Z ghostDirection[4],
      in B ghostEatable[4],
      in Z ghostNormalWeight,
      in Z ghostFacingPacManWight,
      in Z ghostEatableWeight,

      out Z value;

    implementation Math {
        value = 0;
        for i = 1:8
            Z indexY = round(currentY) + i;
            if indexY < 22
                for j = (−i):i
                    Z indexX = round(currentX) + j;
                    if (indexX > 0) && (indexX < 19)
                        for i = 1:4
                            Z xG = round(ghostX(i));
                            Z yG = round(ghostY(i));
                            if (abs (xG − indexX) < 0.1) && (abs(yG −
                                 indexY) < 0.1)
                                Z multiplyer = ghostNormalWeight;
                                if ghostDirection(i) == 0 // Facing towards
                                     PacMan
                                    multiplyer = ghostFacingPacManWight;
                                end
                                if ghostEatable(i)
                                    multiplyer = ghostEatableWeight;
                                end
                                Q dist = sqrt(i∗i + j∗j);
                                value = value + (multiplyer/dist);
                            end
                        end
                    end
                end
            end
        end
    }
}
```

Listing 31: decision.CompareValues

```
package de.rwth.pacman.heithoff2.decision;

// compares all values of the safe directions and takes the maximum
// if the desired direction is blocked (not possible) it tries a direction orthogonal to it
// if those directions are not safe or blocked too, it tries to go the opposite direction
// left is prefered over right and up is prefered over down
component CompareValues {
    ports
      in B topSafe,
      in B bottomSafe,
      in B leftSafe,
      in B rightSafe,
      in Z topValue,
      in Z bottomValue,
      in Z leftValue,
      in Z rightValue,
      in B topPossible,
      in B bottomPossible,
      in B leftPossible,
      in B rightPossible,

      out Z newPacManDirection;

    implementation Math {
        // search maximum
        Z maxValue = −1;
        Z newDirection = 0;
        if topSafe && (topValue > maxValue)
            maxValue = topValue;
        end
        if bottomSafe && (bottomValue > maxValue)
            maxValue = bottomValue;
            newDirection = 1;
        end
        if leftSafe && (leftValue > maxValue)
            maxValue = leftValue;
            newDirection = 2;
        end
        if rightSafe && (rightValue > maxValue)
            newDirection = 3;
        end
        // check whether the desired direction is blocked
        if ((newDirection == 0) && (topPossible == 0)) || ((newDirection == 1) && (
               bottomPossible == 0))
            // pick a direction orthogonal to up/down
            if leftPossible && leftSafe && ((leftValue >= rightValue) || (rightPossible == 0) || (
                   rightSafe == 0))
                newDirection = 2;
            elseif rightPossible && rightSafe
                newDirection = 3;
            // pick the direction opposite to the original direction
            elseif topPossible && topSafe && ((topValue >= bottomValue) || (bottomPossible
                   == 0) || (bottomSafe == 0))
                newDirection = 0;
            else
                newDirection = 1;
            end
        elseif ((newDirection == 2) && (leftPossible == 0)) || ((newDirection == 3) && (
               rightPossible == 0))
            if topPossible && topSafe && ((topValue >= bottomValue) || (bottomPossible ==
                   0) || (bottomSafe == 0))
                newDirection = 0;
            elseif bottomPossible && bottomSafe
                newDirection = 1;
            elseif leftPossible && leftSafe && ((leftValue >= rightValue) || (rightPossible == 0)
                   || (rightSafe == 0))
                newDirection = 2;
            else
                newDirection = 3;
            end
        end
        newPacManDirection = newDirection;
    }
}
```

Listing 32: decision.Decision

```
package de.rwth.pacman.heithoff2.decision;

// Main strategy
component Decision {
    ports
      in B topSafe,
      in B bottomSafe,
      in B leftSafe,
      in B rightSafe,
      in Z topValue,
      in Z bottomValue,
      in Z leftValue,
      in Z rightValue,
      in Q(−1m: 19m) pacManX,
      in Q(0m: 22m) pacManY,
      in Z^{22,19} map,

      out Z newPacManDirection;

    instance CompareValues compareValues; // gives back the desired
        direction
    instance PossibleWays possibleWays; // gives back whether certain
        directions are blocked
    instance VerifyDirection verifyDirection; // prevent stuttering
    instance NextIntersection intersection; // gives back whether an
        intersection (more than 3 non−blocked paths) is reached

    connect pacManX → possibleWays.pacManX, intersection.pacManX;
    connect pacManY → possibleWays.pacManY, intersection.pacManY;
    connect map → possibleWays.map, intersection.map;

    connect topSafe → compareValues.topSafe, verifyDirection.topSafe;
    connect bottomSafe → compareValues.bottomSafe, verifyDirection.
        bottomSafe;
    connect leftSafe → compareValues.leftSafe, verifyDirection.leftSafe;
    connect rightSafe → compareValues.rightSafe, verifyDirection.rightSafe;
    connect topValue → compareValues.topValue;
    connect bottomValue → compareValues.bottomValue;
    connect leftValue → compareValues.leftValue;
    connect rightValue → compareValues.rightValue;
    connect possibleWays.topPossible → compareValues.topPossible,
        verifyDirection.topPossible;
    connect possibleWays.bottomPossible → compareValues.bottomPossible,
        verifyDirection.bottomPossible;
    connect possibleWays.leftPossible → compareValues.leftPossible,
        verifyDirection.leftPossible;
    connect possibleWays.rightPossible → compareValues.rightPossible,
        verifyDirection.rightPossible;

    connect intersection.interSectionReached → verifyDirection.interSection;
    connect compareValues.newPacManDirection → verifyDirection.
        tryDirection;
    connect verifyDirection.newPacManDirection → newPacManDirection;
}
```

Listing 33: decision.NextIntersection

```
package de.rwth.pacman.heithoff2.decision;

// check whether an intersection (3 or more non−blocked paths) is reached

component NextIntersection {
    ports
      in Q(−1m: 19m) pacManX,
      in Q(0m: 22m) pacManY,
      in Zˆ{22,19} map,

      out B interSectionReached;

    implementation Math {
        Z pacX = round(pacManX);
        Z pacY = round(pacManY);
        interSectionReached = 0;
        if (abs(pacManX − pacX) < 0.01) && (abs(pacManY − pacY) <
            0.01)
            pacX = pacX + 1;
            pacY = pacY + 1;
            Zˆ{1,4} xOffSet = [0,0,−1,1];
            Zˆ{1,4} yOffSet = [1,−1,0,0];
            Z newPathsFound = 0;
            for i = 0:3
                Z indexY = 0;
                Z indexX = i;
                Z xOff = xOffSet(indexY, indexX);
                Z yOff = yOffSet(indexY, indexX);
                Q xT = pacX + xOff;
                Q yT = pacY + yOff;

                Z nextTile = map(yT, xT);
                if (nextTile == 0) || (nextTile == 3)
                    newPathsFound = newPathsFound;
                else
                    newPathsFound = newPathsFound + 1;
                    if newPathsFound > 2
                        interSectionReached = 1;
                    end
                end
            end
        end
    }
}
```

Listing 34: decision.PossibleWays

```
package de.rwth.pacman.heithoff2.decision;

// check which directions are not blocked

component PossibleWays {
    ports
      in Q(−1m: 19m) pacManX,
      in Q(0m: 22m) pacManY,
      in Zˆ{22,19} map,

      out B topPossible,
      out B bottomPossible,
      out B leftPossible,
      out B rightPossible;

    implementation Math {
        Qˆ{1,4} xOffSet = [0,0,−0.51,0.51];
        Qˆ{1,4} yOffSet = [0.51,−0.51,0,0];
        topPossible = 0;
        bottomPossible = 0;
        leftPossible = 0;
        rightPossible = 0;

        for i = 0:3
            Z indexX = round(pacManX + xOffSet(0, i)) + 1;
            Z indexY = round(pacManY + yOffSet(0, i)) + 1;
            Z nextTile = map(indexY, indexX);
            if (nextTile != 0) && (nextTile != 3)
                if i == 0
                    bottomPossible = 1;
                elseif i == 1
                    topPossible = 1;
                elseif i == 2
                    leftPossible = 1;
                else
                    rightPossible = 1;
                end
            end
        end
        if abs(pacManX − round(pacManX)) > 0.01
            topPossible = 0;
            bottomPossible = 0;
        elseif abs(pacManY − round(pacManY)) > 0.01
            leftPossible = 0;
            rightPossible = 0;
        end
    }
}
```

Listing 35: decision.VerifyDirection

```
package de.rwth.pacman.heithoff2.decision;

component VerifyDirection {
    ports
      in Z tryDirection,
      in B interSection,
      in B topSafe,
      in B bottomSafe,
      in B leftSafe,
      in B rightSafe,
      in B topPossible,
      in B bottomPossible,
      in B leftPossible,
      in B rightPossible,

      out Z newPacManDirection;

    implementation Math {
        static Z lastDirection = −1;
        newPacManDirection = tryDirection;

        if interSection
            lastDirection = −1;
        elseif ((tryDirection == 0) && (lastDirection == 1)) || ((
             tryDirection == 1) && (lastDirection == 0))
            if leftSafe && leftPossible
                newPacManDirection = 2;
            elseif rightSafe && rightPossible
                newPacManDirection = 3;
            end
            if (tryDirection == 1) && topPossible && topSafe
                newPacManDirection = 0;
            elseif (tryDirection == 0) && bottomPossible && bottomSafe
                newPacManDirection = 1;
            end
        elseif ((tryDirection == 2) && (lastDirection == 3)) || ((
             tryDirection == 3) && (lastDirection == 2))
            if topSafe && topPossible
                newPacManDirection = 0;
            elseif bottomSafe && bottomPossible
                newPacManDirection = 1;
            end
            if (tryDirection == 3) && leftPossible && leftSafe
                newPacManDirection = 2;
            elseif (tryDirection == 2) && rightPossible && rightSafe
                newPacManDirection = 3;
            end
        end

        lastDirection = newPacManDirection;
    }
}
```

Listing 36: decision.TestCompareValues

```
package de.rwth.pacman.heithoff2.decision;

stream TestCompareValues for CompareValues {
    topSafe: 1 tick 1 tick 1 tick 1 tick 1 tick 1;
    bottomSafe: 0 tick 1 tick 1 tick 1 tick 1 tick 1;
    leftSafe: 0 tick 0 tick 0 tick 0 tick 1 tick 1;
    rightSafe: 0 tick 0 tick 0 tick 0 tick 1 tick 1;
    topValue: 0 tick 0 tick 0 tick 0 tick 1 tick 1;
    bottomValue: 0 tick 0 tick 0 tick 1 tick 1 tick 1;
    leftValue: 0 tick 0 tick 0 tick 0 tick 2 tick 1;
    rightValue: 0 tick 0 tick 0 tick 0 tick 1 tick 2;
    topPossible: 1 tick 1 tick 0 tick 1 tick 1 tick 1;
    bottomPossible: 0 tick 1 tick 1 tick 1 tick 1 tick 1;
    leftPossible: 0 tick 0 tick 0 tick 0 tick 1 tick 1;
    rightPossible: 0 tick 0 tick 0 tick 0 tick 1 tick 1;

    newPacManDirection: 0 tick 0 tick 1 tick 1 tick 2 tick 3;
}
```

Listing 37: Flee down

```
package de.rwth.pacman;

stream Test1 for PacManWrapper {
    ghostX: [70cm,88cm,112cm,130cm] tick [70cm,86cm,114cm,130cm] tick
        [70cm,84cm,116cm,130cm] tick [70cm,82cm,118cm,130cm] tick [70cm
        ,80cm,120cm,130cm];
    ghostY: [92cm,70cm,70cm,92cm] tick [94cm,70cm,70cm,94cm] tick [96cm
        ,70cm,70cm,96cm] tick [98cm,70cm,70cm,98cm] tick [100cm,70cm,70
        cm,100cm];
    ghostDirection: [1,2,3,1] tick [1,2,3,1] tick [1,2,3,1] tick [1,2,3,1] tick
        [2,0,0,1];
    ghostEatable: [false, false, false, false] tick [false, false, false, false] tick [
        false, false, false, false] tick [false, false, false, false] tick [false, false,
        false, false];
    ghostEaten: [false, false, false, false] tick [false, false, false, false] tick [
        false, false, false, false] tick [false, false, false, false] tick [false, false,
        false, false];
    pacManX: 70cm tick 70cm tick 70cm tick 70cm tick 70cm;
    pacManY: 130cm tick 132cm tick 134cm tick 136cm tick 138cm;
    pacManEaten: false tick false tick false tick false tick false;
    pacManLives: 3 tick 3 tick 3 tick 3 tick 3;
    pacManScore: 0 tick 0 tick 0 tick 0 tick 0;
    map: ...
    newPacManDirection: 1 tick 1 tick 1 tick 1 tick 1;
}
```

Listing 38: Flee left

```
package de.rwth.pacman;

stream Test2 for PacManWrapper {
    ghostX: [54cm,150cm,170cm,70cm] tick [52cm,150cm,168cm,70cm] tick
        [50cm,150cm,166cm,70cm] tick [48cm,150cm,164cm,70cm];
    ghostY: [210cm,148cm,190cm,172cm] tick [210cm,150cm,190cm,174cm]
        tick [210cm,152cm,190cm,176cm] tick [210cm,154cm,190cm,178cm];
    ghostDirection: [2,1,2,1] tick [2,1,2,1] tick [2,1,2,1] tick [2,1,2,1];
    ghostEatable: [false, false, false, false] tick [false, false, false, false] tick [
        false, false, false, false] tick [false, false, false, false];
    ghostEaten: [false, false, false, false] tick [false, false, false, false] tick [
        false, false, false, false] tick [false, false, false, false];
    pacManX: 150cm tick 150cm tick 148cm tick 146cm;
    pacManY: 130cm tick 132cm tick 134cm tick 136cm;
    pacManEaten: false tick false tick false tick false;
    pacManLives: 3 tick 3 tick 3 tick 3;
    pacManScore: 0 tick 0 tick 0 tick 0;
    map: ...

    newPacManDirection: 0 tick 0 tick 2 tick 2;
}
```

Listing 39: Eat ghosts

```
package de.rwth.pacman;

stream Test3 for PacManWrapper {
    ghostX: [80cm,100cm,55cm,105cm] tick [80cm,100cm,54cm,104cm] tick
        [80cm,100cm,53cm,103cm] tick [80cm,100cm,52cm,102cm] tick [80cm
        ,100cm,51cm,101cm] tick [80cm,100cm,50cm,100cm] tick [79cm,100
        cm,49cm,99cm] tick [78cm,100cm,48cm,98cm] tick [74cm,100cm,47
        cm,97cm] tick [70cm,100cm,46cm,96cm];
    ghostY: [155cm,25cm,10cm,160cm] tick [156cm,24cm,10cm,160cm] tick
        [157cm,23cm,10cm,160cm] tick [158cm,22cm,10cm,160cm] tick [159
        cm,21cm,10cm,160cm] tick [160cm,20cm,10cm,160cm] tick [160cm,19
        cm,10cm,160cm] tick [160cm,18cm,10cm,160cm] tick [160cm,17cm,10
        cm,160cm] tick [160cm,16cm,10cm,160cm];
    ghostDirection: [1,0,2,2] tick [1,0,2,2] tick [1,0,2,2] tick [1,0,2,2] tick
        [1,0,2,2] tick [1,0,2,2] tick [2,0,2,2] tick [2,0,2,2] tick [2,0,2,2] tick
        [2,0,2,2];
    ghostEatable: [1,1,1,1] tick [1,1,1,1] tick [1,1,1,1] tick [1,1,1,1] tick
        [1,1,1,1] tick [1,1,1,1] tick [1,1,1,1] tick [0,1,1,1] tick [0,1,1,1] tick
        [0,1,1,1];
    ghostEaten: [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0]
         tick [0,0,0,0] tick [0,0,0,0] tick [1,0,0,0] tick [1,0,0,0] tick [1,0,0,0];
    pacManX: 56cm tick 58cm tick 60cm tick 62cm tick 64cm tick 66cm
        tick 68cm tick 70cm tick 72cm tick 74cm;
    pacManY: 160cm tick 160cm tick 160cm tick 160cm tick 160cm tick
        160cm tick 160cm tick 160cm tick 160cm tick 160cm;
    pacManEaten: 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0
        tick 0;
    pacManLives: 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3
        tick 3;
    pacManScore: 250 tick 250 tick 250 tick 260 tick 260 tick 260 tick 260
        tick 310 tick 320 tick 320;
    map: ...

    newPacManDirection: 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3
        tick 3 tick 3;
}
```

Listing 40: Eat biscuits

```
package de.rwth.pacman;

stream Test4 for PacManWrapper {
    ghostX: [44cm,166cm,140cm,60cm] tick [46cm,164cm,140cm,60cm] tick
        [48cm,162cm,140cm,60cm] tick [50cm,160cm,140cm,60cm] tick [52cm
        ,158cm,142cm,62cm] tick [54cm,156cm,144cm,64cm] tick [56cm,154
        cm,146cm,66cm] tick [58cm,152cm,148cm,68cm] tick [60cm,150cm
        ,150cm,70cm] tick [62cm,148cm,152cm,72cm];
    ghostY: [10cm,40cm,46cm,54cm] tick [10cm,40cm,44cm,56cm] tick [10cm
        ,40cm,42cm,58cm] tick [10cm,40cm,40cm,60cm] tick [10cm,40cm,40
        cm,60cm] tick [10cm,40cm,40cm,60cm] tick [10cm,40cm,40cm,60cm]
        tick [10cm,40cm,40cm,60cm] tick [10cm,40cm,40cm,60cm] tick [10
        cm,40cm,40cm,60cm];
    ghostDirection: [3,2,0,1] tick [3,2,0,1] tick [3,2,0,1] tick [3,2,0,1] tick
        [3,2,3,3] tick [3,2,3,3] tick [3,2,3,3] tick [3,2,3,3] tick [3,2,3,3] tick
        [3,2,3,3];
    ghostEatable: [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick
        [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick
        [0,0,0,0];
    ghostEaten: [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0]
         tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0];
    pacManX: 120cm tick 120cm tick 120cm tick 120cm tick 122cm tick
        124cm tick 126cm tick 128cm tick 130cm tick 132cm;
    pacManY: 166cm tick 164cm tick 162cm tick 160cm tick 160cm tick
        160cm tick 160cm tick 160cm tick 160cm tick 160cm;
    pacManEaten: 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0
        tick 0;
    pacManLives: 2 tick 2 tick 2 tick 2 tick 2 tick 2 tick 2 tick 2 tick 2
        tick 2;
    pacManScore: 260 tick 260 tick 260 tick 260 tick 270 tick 270 tick 270
        tick 270 tick 270 tick 280;
    map: ...

    newPacManDirection: 0 tick 0 tick 0 tick 3 tick 3 tick 3 tick 3 tick 3
        tick 3 tick 3;
}
```

Listing 41: Eat coin

```
package de.rwth.pacman;

stream Test5 for PacManWrapper {
    ghostX: [40cm,60cm,86cm,140cm] tick [40cm,60cm,84cm,140cm] tick [40
        cm,60cm,82cm,140cm] tick [40cm,60cm,80cm,140cm] tick [40cm,62
        cm,78cm,142cm] tick [40cm,64cm,76cm,144cm] tick [40cm,66cm,74
        cm,146cm] tick [40cm,68cm,72cm,148cm] tick [40cm,70cm,70cm,150
        cm] tick [40cm,72cm,68cm,152cm];
    ghostY: [54cm,174cm,80cm,174cm] tick [56cm,176cm,80cm,176cm] tick
        [58cm,178cm,80cm,178cm] tick [60cm,180cm,80cm,180cm] tick [62cm
        ,180cm,80cm,180cm] tick [64cm,180cm,80cm,180cm] tick [66cm,180
        cm,80cm,180cm] tick [68cm,180cm,80cm,180cm] tick [70cm,180cm,80
        cm,180cm] tick [72cm,180cm,80cm,180cm];
    ghostDirection: [1,1,2,1] tick [1,1,2,1] tick [1,1,2,1] tick [1,1,2,1] tick
        [1,3,2,3] tick [1,3,2,3] tick [1,3,2,3] tick [1,3,2,3] tick [1,3,2,3] tick
        [1,3,2,3];
    ghostEatable: [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick
        [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick
        [0,0,0,0];
    ghostEaten: [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0]
         tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0] tick [0,0,0,0];
    pacManX: 34cm tick 32cm tick 30cm tick 28cm tick 26cm tick 24cm
        tick 22cm tick 20cm tick 20cm tick 20cm;
    pacManY: 180cm tick 180cm tick 180cm tick 180cm tick 180cm tick
        180cm tick 180cm tick 180cm tick 178cm tick 176cm;
    pacManEaten: 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0 tick 0
        tick 0;
    pacManLives: 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3 tick 3
        tick 3;
    pacManScore: 200 tick 200 tick 200 tick 210 tick 210 tick 210 tick 210
        tick 210 tick 220 tick 220;
    map: ...

    newPacManDirection: 2 tick 2 tick 2 tick 2 tick 2 tick 2 tick 2 tick 2
        tick 0 tick 0;
}
```

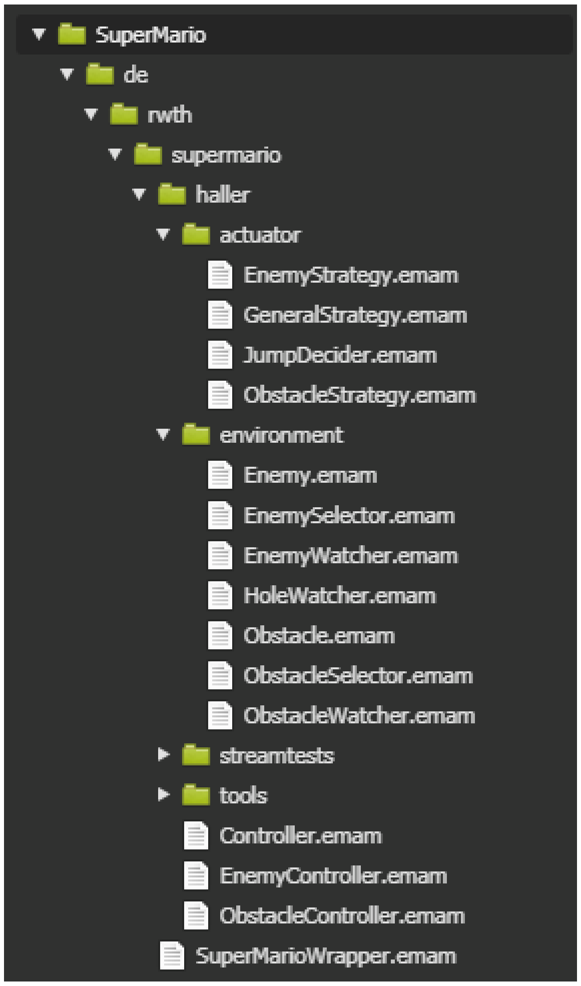


Figure C.26: Supermario package outline

Listing 42: SuperMarioWrapper

```
package de.rwth.supermario;

import de.rwth.supermario.haller.Controller;

component SuperMarioWrapper {
    ports
        in Z^{1,2} marioPosition,
        in Z^{1,2} marioVelocity,
        in Z marioHeight,
        in Z^{5,2} nextEnemyPositions,
        in Z^{5,2} nextObstaclePositions,
        in Z nextHole,
        in Z^{5,2} nextLootCrates,
        in Q tickSize,
        in Z marioResting,
        out Z(−1 : 1 : 1) marioDirection,
        out Z marioJump,
        out Z marioDown,
        out Z marioShoot;


    //Replace this with your own custom controller
    instance Controller controller;

    connect marioPosition → controller.marioPosition;
    connect marioVelocity → controller.marioVelocity;
    connect marioHeight → controller.marioHeight;
    connect nextEnemyPositions → controller.nextEnemyPositions;
    connect nextObstaclePositions → controller.nextObstaclePositions;
    connect nextHole → controller.nextHole;
    connect nextLootCrates → controller.nextLootCrates;
    connect tickSize → controller.tickSize;
    connect marioResting → controller.marioResting;

    connect controller.marioJump → marioJump;
    connect controller.marioDirection → marioDirection;
    connect controller.marioDown → marioDown;
    connect controller.marioShoot → marioShoot;

}
```

Listing 43: Controller

```
package de.rwth.supermario.haller;
import de.rwth.supermario.haller.tools.OrRelation_3;
import de.rwth.supermario.haller.actuator.GeneralStrategy;
import de.rwth.supermario.haller.actuator.JumpFilter;
import de.rwth.supermario.haller.EnemyController;
import de.rwth.supermario.haller.ObstacleController;
component Controller {
    ports
        in Z^{1,2} marioPosition,
        in Z^{1,2} marioVelocity,
        in Z marioHeight,
        in Z^{5,2} nextEnemyPositions,
        in Z^{5,2} nextObstaclePositions,
        in Z nextHole,
        in Z^{5,2} nextLootCrates,
        in Q tickSize,
        in Z marioResting,
        out Z(−1 : 1 : 1) marioDirection,
        out Z marioJump,
        out Z marioDown,
        out Z marioShoot;

    //Sub−Controllers to keep this file clean and enhance overall modularity
    //This one deals with the enemies
    instance EnemyController enemyController;
    connect nextEnemyPositions → enemyController.nextEnemyPositions;
    //This one deals with pipes, staircases and holes
    instance ObstacleController obstController;
    connect nextObstaclePositions → obstController.nextObstaclePositions;
    connect nextHole → obstController.holeDistance;
    //This Strategy determines the overall strategy.
    //Right now this only consists of a stuck−detection and constantly going
        to the right
    instance GeneralStrategy genStrategy;
    connect tickSize → genStrategy.tickSize;
    connect marioPosition→ genStrategy.marioPosition;
    //The jumpAdvice result of the two controllers and the general strategy
        are combined via or
    instance OrRelation_3 orR;
    connect obstController.jumpAdvice → orR.input[1];
    connect enemyController.jumpAdvice → orR.input[2];
    connect genStrategy.jumpAdvice → orR.input[3];
    //Checked vor validity
    instance JumpFilter jumpFilter;
    connect orR.output → jumpFilter.jumpAdvice;
    connect marioResting → jumpFilter.marioResting;
    //And forwarded
    connect jumpFilter.marioJump → marioJump;
    //The general strategy is currently the only entity making decisions on
        direction and crouching
    connect genStrategy.directionAdvice → marioDirection;
    connect genStrategy.crouchAdvice → marioDown;
}
```

Listing 44: EnemyController

```
package de.rwth.supermario.haller;

import de.rwth.supermario.haller.tools.GetIndexes;
import de.rwth.supermario.haller.environment.Enemy;
import de.rwth.supermario.haller.environment.EnemySelector;
import de.rwth.supermario.haller.actuator.EnemyStrategy;

component EnemyController {
    ports
        in Z^{5,2} nextEnemyPositions,

        out Z(−1 : 1 : 1) dirAdvice,
        out Z jumpAdvice,
        out Z shootAdvice,
        out Z crouchAdvice;

    //Helper component to make the code shorter
    instance GetIndexes enemyIndexes;

    //These selectors select the according x and y positions from the Array
    instance EnemySelector enemySelectors[5];
    connect enemyIndexes.index[:] → enemySelectors[:].index;

    connect nextEnemyPositions → enemySelectors[1].nextEnemyPositions,
            enemySelectors[2].nextEnemyPositions,
            enemySelectors[3].nextEnemyPositions,
            enemySelectors[4].nextEnemyPositions,
            enemySelectors[5].nextEnemyPositions;

    instance Enemy enemies[5];
    connect enemySelectors[:].x → enemies[:].distX;
    connect enemySelectors[:].y → enemies[:].distY;

    //The values are forwarded into the strategy
    instance EnemyStrategy enemyStrat;
    connect enemySelectors[:].x → enemyStrat.enemyDistsX[:];
    connect enemySelectors[:].y → enemyStrat.enemyDistsY[:];
    connect enemies[:].velX → enemyStrat.enemyVelX[:];
    connect enemies[:].velY → enemyStrat.enemyVelY[:];

    //The advice ist passed back
    connect enemyStrat.jumpAdvice → jumpAdvice;
}
```

Listing 45: ObstacleController

```
package de.rwth.supermario.haller;

import de.rwth.supermario.haller.tools.GetIndexes;
import de.rwth.supermario.haller.environment.Obstacle;
import de.rwth.supermario.haller.environment.ObstacleSelector;
import de.rwth.supermario.haller.actuator.ObstacleStrategy;

component ObstacleController {
    ports
        in Zˆ{5,2} nextObstaclePositions,
        in Z holeDistance,

        out Z(−1 : 1 : 1) dirAdvice,
        out Z jumpAdvice,
        out Z shootAdvice,
        out Z crouchAdvice;

    //Helper component to make the code shorter
    instance GetIndexes obstIndexes;

    //These selectors select the according x and y positions from the Array
    instance ObstacleSelector obstacleSelectors[5];
    connect obstIndexes.index[:] → obstacleSelectors[:].index;

    connect nextObstaclePositions → obstacleSelectors[1].
        nextObstaclePositions,
            obstacleSelectors[2].nextObstaclePositions,
            obstacleSelectors[3].nextObstaclePositions,
            obstacleSelectors[4].nextObstaclePositions,
            obstacleSelectors[5].nextObstaclePositions;

    instance Obstacle obstacles[5];
    connect obstacleSelectors[:].x → obstacles[:].distX;
    connect obstacleSelectors[:].y → obstacles[:].distY;

    //The values are forwarded into the strategy
    instance ObstacleStrategy obstStrat;
    connect obstacleSelectors[:].x → obstStrat.obstDistsX[:];
    connect obstacleSelectors[:].y → obstStrat.obstDistsY[:];
    connect holeDistance → obstStrat.holeDistance;

    //The advice ist passed back
    connect obstStrat.jumpAdvice → jumpAdvice;
}
```

Listing 46: actuator.GeneralStrategy

```
package de.rwth.supermario.haller.actuator;

component GeneralStrategy {
    ports
        in Z^{1,2} marioPosition,
        in Q tickSize,

        out Z jumpAdvice,
        out Z crouchAdvice,
        out Z directionAdvice;

    implementation Math {
        //Wait one seconds before being "stuck"
        Z maxTicks = 0.5 * tickSize;

        static Z ticksOnSamePosition = 0;
        static Z oldXPos = −1;

        if oldXPos == marioPosition(1,1)
            ticksOnSamePosition = ticksOnSamePosition + 1;
        else
            oldXPos = marioPosition(1,1);
            ticksOnSamePosition = 0;
        end

        if(ticksOnSamePosition > maxTicks)
            jumpAdvice = 1;
        else
            jumpAdvice = 0;
        end

        directionAdvice = 1;

        crouchAdvice = 0;
    }
}
```

Listing 47: actuator.EnemyStrategy

```
package de.rwth.supermario.haller.actuator;

import de.rwth.supermario.haller.tools.OrRelation_5;
import de.rwth.supermario.haller.environment.Enemy;
import de.rwth.supermario.haller.environment.EnemyWatcher;

component EnemyStrategy {
    ports
        in Z enemyDistsX[5],
        in Z enemyDistsY[5],
        in Z enemyVelX[5],
        in Z enemyVelY[5],
        in Z^{1,2} marioPosition,

        out Z jumpAdvice,
        out Z directionAdvice;

    //Every EnemyWatcher watches a single Enemy
    instance EnemyWatcher enemyWatchers[5];
    connect enemyDistsX[:] -> enemyWatchers[:].EnemyDistX;
    connect enemyDistsY[:] -> enemyWatchers[:].EnemyDistY;
    connect enemyVelX[:] -> enemyWatchers[:].EnemyVelocityX;
    connect enemyVelY[:] -> enemyWatchers[:].EnemyVelocityY;

    //The output of all Watchers is combined via an or-relation.
    instance OrRelation_5 orR;
    connect enemyWatchers[:].inJumpRange -> orR.input[:];

    //The result is forwarded
    connect orR.output -> jumpAdvice;
}
```

Listing 48: actuator.ObstacleStrategy

```
package de.rwth.supermario.haller.actuator;

import de.rwth.supermario.haller.tools.OrRelation_2;
import de.rwth.supermario.haller.tools.OrRelation_5;
import de.rwth.supermario.haller.environment.Obstacle;
import de.rwth.supermario.haller.environment.ObstacleWatcher;

import de.rwth.supermario.haller.environment.HoleWatcher;

component ObstacleStrategy {
    ports
        in Z obstDistsX[5],
        in Z obstDistsY[5],
        in Z holeDistance,

        out Z jumpAdvice,
        out Z directionAdvice;

    //Every ObstacleWatcher watches a single Obstacle.
    //Obstacles are pipes and staircase blocks.
    instance ObstacleWatcher obstacleWatchers[5];
    connect obstDistsX[:] → obstacleWatchers[:].ObstacleDistX;
    connect obstDistsY[:] → obstacleWatchers[:].ObstacleDistY;

    //The output of all Watchers is combined via an or−relation.
    instance OrRelation_5 orR_5;
    connect obstacleWatchers[:].inJumpRange → orR_5.input[:];

    //The HoleWatcher watches the distance to the next hole.
    instance HoleWatcher holeWatch;
    connect holeDistance → holeWatch.holeDistance;

    //Finally, the result from the watchers are combined via or
    instance OrRelation_2 orR_2;
    connect holeWatch.inJumpRange → orR_2.input[1];
    connect orR_5.output → orR_2.input[2];

    //This results in the final advice for obstacle handling
    connect orR_2.output → jumpAdvice;
}
```

Listing 49: actuator.JumpFilter

```
package de.rwth.supermario.haller.actuator;

component JumpFilter {
    ports
        in Z jumpAdvice,
        in Z marioResting,

        out Z marioJump;

    implementation Math{
        //Once Mario lands, he needs to stop "jumping" for once, since the
        //simulator only jumps once if the jump key is pressed.
        static Z marioAlreadyRestedOnce = 0;

        if(marioResting == 0) //We are in the air
            if(jumpAdvice==1) //Update the "we already rested"-flag
                marioAlreadyRestedOnce = 0;
            else
                marioAlreadyRestedOnce = 1;
            end
            marioJump = jumpAdvice;
        else
            if(marioAlreadyRestedOnce == 1)
                marioJump = jumpAdvice;
            else
                marioAlreadyRestedOnce = 1;
                marioJump = 0;
            end
        end
    }
}
```

Listing 50: environment.Enemy

```
package de.rwth.supermario.haller.environment;

import de.rwth.supermario.haller.tools.GetVelocity;

component Enemy {
    ports
        in Z distX,
        in Z distY,

        out Z velX,
        out Z velY;

    instance GetVelocity velocity;
    connect distX → velocity.distX;
    connect distY → velocity.distY;
    connect velocity.velX → velX;
    connect velocity.velY → velY;
}
```

Listing 51: environment.EnemySelector

```
package de.rwth.supermario.haller.environment;

component EnemySelector {
    ports
        in Zˆ{5,2} nextEnemyPositions,
        in Z index,

        out Z x,
        out Z y;

    implementation Math {
        x = nextEnemyPositions(index,1);
        y = nextEnemyPositions(index,2);
    }
}
```

Listing 52: environment.EnemyWatcher

```
package de.rwth.supermario.haller.environment;

component EnemyWatcher {
    ports
        in Z EnemyDistX,
        in Z EnemyDistY,
        in Z EnemyVelocityX,
        in Z EnemyVelocityY,

        out Z movesTowardsPlayer,
        out Z inJumpRange;

    implementation Math {
        //Empirical distance values
        Z jumpRangeX = 200;

        //Enemy in Jumprange, and not above, not undefined, stopping to
            jump while we are over it, so we don't jump much too far.
        if((abs(EnemyDistX) < jumpRangeX) && (EnemyDistY > −64) &&
             (EnemyDistX != −1) && (EnemyDistX > 45))
            inJumpRange = 1;
        else
            inJumpRange = 0;
        end


        //Enemy moving in the opposite direction of the direction to it from
            mario
        if((EnemyVelocityX > 0) != (EnemyDistX > 0))
            movesTowardsPlayer = 1;
        else
            movesTowardsPlayer = 0;
        end
    }
}
```

Listing 53: environment.HoleWatcher

```
package de.rwth.supermario.haller.environment;

component HoleWatcher {
    ports
        in Z holeDistance,

        out Z inJumpRange;

    implementation Math {
        //Empirical distance values
        Z jumpRangeX = 128;

        if((abs(holeDistance) < jumpRangeX) && (holeDistance != −1))
            inJumpRange = 1;
        else
            inJumpRange = 0;
        end
    }
}
```

Listing 54: environment.Obstacle

```
package de.rwth.supermario.haller.environment;

import de.rwth.supermario.haller.tools.GetVelocity;

component Obstacle {
    ports
        in Z distX,
        in Z distY,

        out Z velX,
        out Z velY;

    instance GetVelocity velocity;

    connect distX → velocity.distX;
    connect distY → velocity.distY;
    connect velocity.velX → velX;
    connect velocity.velY → velY;
}
```

Listing 55: environment.ObstacleSelector

```
package de.rwth.supermario.haller.environment;

component ObstacleSelector {
    ports
        in Zˆ{5,2} nextObstaclePositions,
        in Z index,

        out Z x,
        out Z y;

    implementation Math {
        x = nextObstaclePositions(index,1);
        y = nextObstaclePositions(index,2);
    }
}
```

Listing 56: environment.ObstacleWatcher

```
package de.rwth.supermario.haller.environment;

component ObstacleWatcher {
    ports
        in Z ObstacleDistX,
        in Z ObstacleDistY,

        out Z inJumpRange;

    implementation Math {
        //Empirical distance values
        Z jumpRangeX = 96;

        if((abs(ObstacleDistX) < jumpRangeX) && (ObstacleDistX != −1))
            inJumpRange = 1;
        else
            inJumpRange = 0;
        end
    }
}
```

Listing 57: tools.GetVelocity

```
package de.rwth.supermario.haller.tools;


component GetVelocity {
    ports //x,y
        in Z distX,
        in Z distY,

        out Z velX,
        out Z velY;

    implementation Math {
        static Z oldDistX = −1;
        static Z oldDistY = −1;

        //Calculate velocity (distance / ticklength)
        if(oldDistX != −1)
            velX = distX − oldDistX;
            velY = distY − oldDistY;
            oldDistX = distX;
            oldDistY = distY;

        else
            velX = −1;
            velY = −1;
        end
    }
}
```

Listing 58: tools.GetNearest

```
package de.rwth.supermario.haller.tools;


component GetNearest {
    ports //x,y
        in Z^{5,2} array,

        out Z nearestDistX,
        out Z nearestDistY,
        out Z index;

    implementation Math {
        //Implement Bubblesort?
        for i = 1:5
            //Z ecDist = sqrt(distX * distX + distY * distY); //Euclidic
                Distance

            if(array(i,1) > 0 )
                nearestDistX = array(i,1);
                nearestDistY = array(i,2);
                index = i;
            else
                nearestDistX = −1;
                nearestDistY = −1;
                index = −1;
            end
        end
    }
}
```

Appendix D. Supermario Stream Test Code

Listing 59: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_Evade for EnemyWatcher {
    EnemyDistX: 200 tick 100 tick 75;
    EnemyDistY: 0 tick 0 tick 0;
    EnemyVelocityX: −10 tick −10 tick −10;
    EnemyVelocityY: 0 tick 0 tick 0;

    movesTowardsPlayer: 1 tick 1 tick 1;
    inJumpRange: 0 tick 0 tick 1;
}
```

Listing 60: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_FromAbove for EnemyWatcher {
    EnemyDistX: 200 tick 100 tick 5;
    EnemyDistY: 128 tick 128 tick 32;
    EnemyVelocityX: −10 tick −10 tick −10;
    EnemyVelocityY: 0 tick 0 tick 0;

    movesTowardsPlayer: 1 tick 1 tick 1;
    inJumpRange: 0 tick 0 tick 0;
}
```

Listing 61: Enemy watcher stream test

```
package de.rwth.supermario.haller.environment;

stream Env_EnemyWatcher_FromAbove for EnemyWatcher {
    EnemyDistX: −1 tick;
    EnemyDistY: −1 tick;
    EnemyVelocityX: 0 tick;
    EnemyVelocityY: 0 tick;

    movesTowardsPlayer: 0 tick;
    inJumpRange: 0 tick;
}
```

Listing 62: Obstacle watcher stream test

```
package de.rwth.supermario.haller.environment;
stream Env_ObstacleWatcher for ObstacleWatcher {
  ObstacleDistX: 200 tick 100 tick 75 tick 50 tick 25 tick 0;
  ObstacleDistX: 0 tick 0 tick 0 tick 25 tick 50 tick 75;

  inJumpRange: 0 tick 0 tick 1 tick 1 tick 1 tick 0;
}
```

Listing 63: Hole watcher stream test

```
package de.rwth.supermario.haller.environment;
stream Env_ObstacleWatcher for ObstacleWatcher {
    holeDistance: 200 tick 100 tick 10 tick 0 tick 1200;

    inJumpRange: 0 tick 0 tick 1 tick 1 tick 0;
}
```

References

[1] M. von Wenckstern, Embeddedmontiarc demo video.
URL `https://www.youtube.com/watch?v=VTKSWwWp-kg`

[2] M. Heithoff, Pacman model.
URL `https://embeddedmontiarc.github.io/SuperMario/PacMan/`

[3] P. Haller, Supermario model.
URL `https://embeddedmontiarc.github.io/SuperMario/supermario/simulation.html`

[4] V. Bertram, S. Maoz, J. O. Ringert, B. Rumpe, M. von Wenckstern, Component and connector views in practice: An experience report, ACM/IEEE International Conference on Model Driven Engineering Languages and Systems 20.

[5] Filippo Grazioli, Evgeny Kusmenko, Alexander Roth, Bernhard Rumpe, Michael von Wenckstern, Simulation framework for executing component and connector models of self-driving vehicles, Proceedings of MODELS 2017. Workshop EXE, Austin, CEUR 2019, Sept. 2017.

[6] OMG, Sysml.
URL `http://www.omgsysml.org`

[7] SAE, Architecture analysis and design language.
URL `http://www.aadl.info/`

[8] Mathworks, Simulink.
URL `https://de.mathworks.com/products/simulink.html`

[9] N. Instruments, Labview.
URL `http://www.ni.com/de-de/shop/labview.html`

[10] K. Hölldobler, B. Rumpe, MontiCore 5 Language Workbench Edition 2017, Aachener Informatik-Berichte, Software Engineering, Band 32, Shaker Verlag, 2017.
URL `http://www.se-rwth.de/phdtheses/MontiCore-5-Language-Workbench-Edition-2017.pdf`

[11] E. Kusmenko, A. Roth, B. Rumpe, M. von Wenckstern, Modeling Architectures of Cyber-Physical Systems, in: European Conference on Modelling Foundations and Applications (ECMFA'17), LNCS 10376, Springer, 2017, pp. 34–50.
URL `http://www.se-rwth.de/publications/Modeling-Architectures-of-Cyber-Physical-Systems.pdf`

[12] S. Hillemacher, S. Kriebel, E. Kusmenko, M. Lorang, B. Rumpe, A. Sema, G. Strobl, M. von Wenckstern, Model-Based Development of Self-Adaptive Autonomous Vehicles using the SMARDT Methodology, in: Proceedings

of the 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD'18), SciTePress, 2018, pp. 163 – 178.

[13] C. of Software Engineering RWTH Aachen University, Homepage of the chair of software engineering at rwth aachen university.
URL `http://se-rwth.de`

[14] A. Mokhtarian, Monticar: 3d modeling using embeddedmontiarcmath (2018).

[15] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering.

[16] D. Harvey, Html5 pacman.
URL `https://demo.embeddedmontiarc.com/pacman2/`

[17] J. Goldberg, Fullscreenmario, html5 browser game.
URL `http://www.joshuakgoldberg.com/FullScreenMario/Source/`

[18] E. Kusmenko, B. Rumpe, S. Schneiders, M. von Wenckstern, Highly-Optimizing and Multi-Target Compiler for Embedded System Models: C++ Compiler Toolchain for the Component and Connector Language EmbeddedMontiArc, in: Conference on Model Driven Engineering Languages and Systems (MODELS'18), IEEE, 2018.

[19] E. Team, Embeddedmontiarc documentation.
URL `https://github.com/EmbeddedMontiArc/Documentation`