8 Applying the proposed domain specific language

This chapter presents the final part of our case study – the design of activities for *Pervasive Word Search*, using the proposed DSL to specify all activities in this game, in both text and visual format. This chapter also presents other examples related to a multi-player game (*The Audio Flashlight 5*) in Section 8.2. At the end (Section 8.3), this chapter explains how using diagrams was useful for developing the prototypes.

8.1 Pervasive Word Search

In *Pervasive Word Search*, there are four primary activities: *Capture color*, *Interact with wireless zone*, *Interact with dark zone*, and *Interact with open zone*. There is one secondary activity: *End play session*. Figure 8.1 illustrates how these activities are related, using the elements defined in the previous chapter.

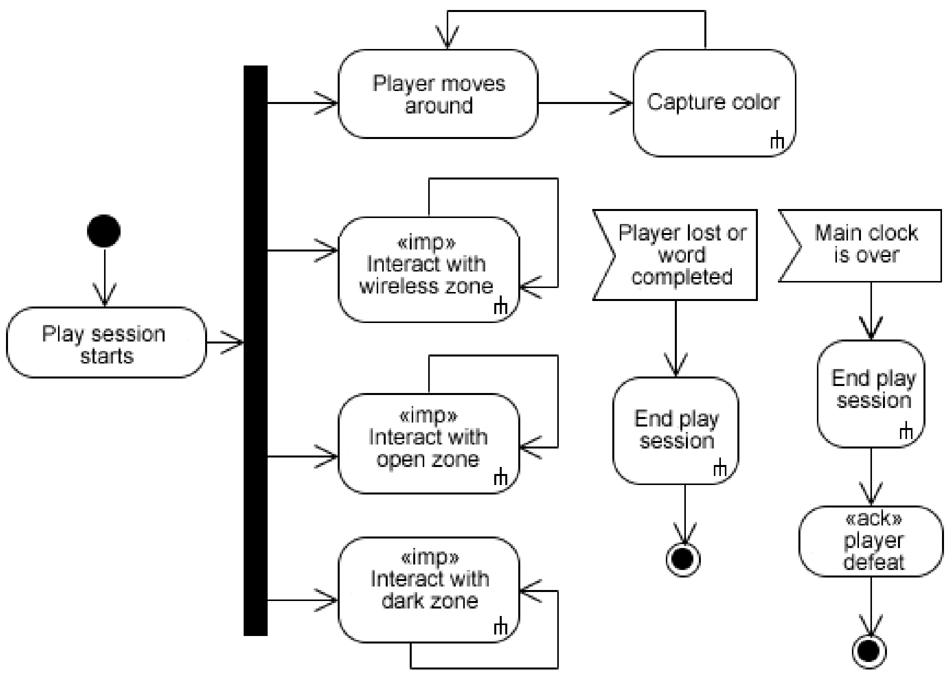
Player moves around
Capture color
Play session starts
«imp» Interact with wireless zone
«imp» Interact with open zone
«imp» Interact with dark zone
Player lost or word completed
Main clock is over
End play session
End play session
«ack» player defeat

Figure 8.1: Relationships among activities in *Pervasive Word Search*

Figure 8.1 describes that all activities occur in parallel, and that *Capture color* is the one that has *explicit* style. The other ones have *implicit* style, and start when a play session starts.

We first specify the sensors and actuators that this game requires. Table 8.1 summarizes this information[93].

Sensors	Actuators
Touch screen, Bluetooth, WiFi, Camera, Light sensor, GPS	Vibration motors, Display

Table 8.1: Sensors and actuators required by *Pervasive Word Search*

Handling events related to smartphones is important as they might interrupt the game session. As in this game this event handling is the same for all activities, we separate them into their own section in the game design document file, as Figure 8.2 illustrates:

```
Alternative flows for smartphone events (common for all activities):
  •  Low battery:
        Perform Handle low battery event.

  •  Incoming call or message:
        Perform Handle incoming call or message


-- Smartphone Event 1: Handle low battery event

Scope                         Support
Primary Actors                Player, OS
Sensors                       -
Actuators                     Display
Interaction granularity       -
Control                       -

Default flow
  1. OS notifies game about low-battery level
     •  ack (display):  low battery warning for the player

  2. Game ends application


-- Smartphone event 2: Handle incoming call or message

Scope                         Support
Primary Actors                Player, OS
Sensors                       -
Actuators                     Display
Interaction granularity       -
Control                       -

Default flow
  1. Player receives an incoming call or incoming SMS/MMS
  2. Game enters "pause state"
```

Figure 8.2: Text specification of main smartphone events

93 The specification of *Pervasive Word Search* uses elements of the enhanced game design template that Appendix D presents.

The remaining of this section presents all activities of *Pervasive Word Search*.

8.1.1 Capture color

In the *Capture color* activity, the player wanders around the physical environment to find the colors he needs to complete the *target word* (the word the game has chosen for the player). The player then captures a color with the device camera. This example illustrates using the following concepts:

- Activity with explicit style, and discrete granularity;
- Acknowledgments (explicit and implicit style);
- Generic events;
- Invoking other activities;
- Branching flows;
- Ending alternative flows (without finishing the whole activity).

Figure 8.3 illustrates the activity diagram.

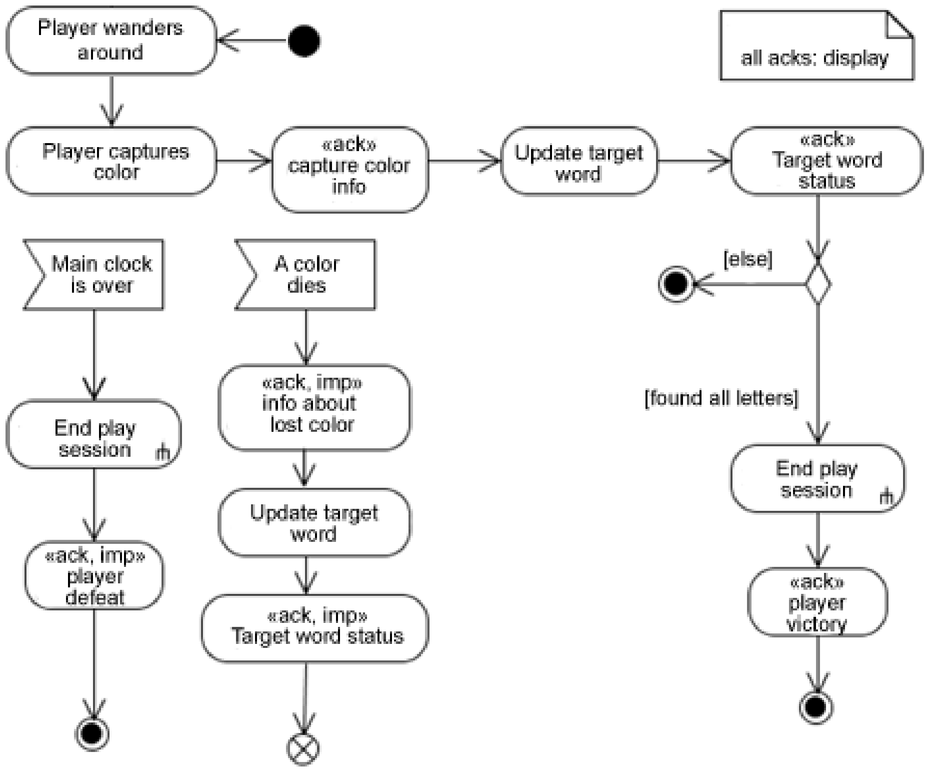

Figure 8.3: "*Capture color*" activity diagram

The activity starts when the player wanders around and captures a color with the device. The game evaluates the color, informing the player about the results (through *acks*), and updates the target word. If all letters have been found, the game ends the play session. This diagram illustrates invoking another activity (*End play session*) and generic events interrupting or affecting the activity flow (*Main clock is over*, *A color dies*).

Notice that when the event *A color dies* occurs, the activity does not finish when the flow started by the event finishes. This event is important because when a color "dies" (*i.e.* its life span time expires), the player loses the letters associated with the color, possibly affecting the status of the target word. When the event *Main clock is over* occurs, its flow finishes the whole activity.

In this example all acknowledgments are delivered through the device display, hence the note at the top of the diagram indicating this information to make the diagram cleaner. Figure 8.4 illustrates the textual specification of this activity.

Scope	Primary
Primary Actors	Player
Sensors	Camera
Actuators	Display
Interaction granularity	Discrete
Control	Explicit

Default flow
1. Player positions camera to focus the target object and captures color
   - **ack (display):** captured color
2. Game identifies the color and uses the color name to update the target word state
   - **ack (display):** current target word state

Alternative flows

2a. All letters of target word have been captured:
1. Perform ***End play session***
   - **ack (display):** announces player victory

*a. Main game clock runs out:
1. Perform ***End play session***
   - **ack (display):** announces player defeat
2. **EOA**

*a. A color dies:
1. Update target word state
   - **ack (display):** info about lost color
   - **ack (display):** current target word state

Uncertainty handling policy
- Hide
  - ▫ The colors are represented with a limited set of options. Similar "colors" then are grouped with the same name (*e.g.* all variations of "red" are considered as "red") - "discretization technique".

Figure 8.4: Text specification of the *Capture color* activity

The first part of this specification lists the involved actors (in this case, just the player), the sensors and actuators that this activity uses, the interaction granularity, and activity control type. The interaction granularity is discrete because this activity concerns capturing a specific color using the device camera from a set of finite possibilities. The control is explicit, as the player has to capture the color through a direct (conscious) command.

The *default flow* is composed of two steps. In this example, each step has an associated acknowledgment, which the game delivers as consequences of the correspondent player interactions. For example, the item *"*`ack (display): captured color`*"* in step `1` defines that the game should inform the player about the color just captured through the device display. The step numbering corresponds to the sequence that those steps occur.

This activity has three alternative flows, indicated by `2a`, and the two `*a` flows. The notation `2a` means that the alternative path started in `2a` is an alternative to step `2` in the default flow. The condition to use this flow is expressed as "all letters of target word have been captured". The conditions end with a colon. In this case, the flow defined in `2a` replaces step `2`. Notice that in this example, it ends with **`EOA`** (end of activity) – this notation indicates that the whole activity finishes at that point, if this alternative flow is invoked. Otherwise, the activity would go back to the default flow, resuming it for the next step (this would be step `3`, which does not exist in this example). The **`EOA`** notation should be used if it is not clear or obvious where (*e.g.* at which step) the activity ends.

The alternative flows specified in the two `*a` flows represent alternative paths that may occur during any step in the default flow. Each one represents a distinct (generic) event: *Main game clock runs out* and *A color dies*. The alternative path *Main game clock runs out* is invoked when the game clock runs out, interrupting the whole activity. In this case, the alternative path invokes another activity (*End play session*) and terminates *Capture Color*, which is indicated with the **`EOA`** notation. However, the alternative flow *A color dies* does not terminate the whole activity.

The interaction granularity is *discrete* because this activity concerns capturing a specific color (using a target box) from a set of finite possibilities. The control is *explicit*, as the player has to capture the color through a direct command.

The *Uncertainty handling policy* item indicates how the game handles uncertainties related to the camera. In this example, the activity should use the hide strategy, by collapsing several possible (infinite) “colors” into a set of finite possibilities. For example, several colors that could be identified as “red” are mapped to a single “red” color value, eliminating the (continuous) nature of color.

8.1.2 Interact with wireless zone

In this activity, the player wanders around in the physical world while the game searches for Bluetooth and WiFi devices in the background. The game uses letters from device names (defined as “wireless set”) to complete the target word name. This is an implicit-styled activity – all acknowledgments are implicit. Figure 8.5 illustrates the activity diagram for *Interact with wireless zone*.

This example illustrates using the following concepts:

- Activity with implicit style, and continuous granularity;
- Acknowledgments (implicit style);
- Generic and interaction events;
- Invoking other activities;
- Branching and merging flows;
- Concurrent flows.

This activity starts automatically when the play session starts. The game initiates searching for Bluetooth and WiFi devices in the background, while the player keeps wandering around somewhere. As the player has no control on this process, this is an implicit-styled action. When the search is over, the game produces interaction events and their correspondent (implicit) acknowledgment to the player. This implicit interaction may affect the target word state, either completing it, or making some letters disappear. This activity ends if the target word is completed (the player wins), or if the main clock runs out (the player loses). In either case the acknowledgments have implicit style, as those events were not caused by conscious player actions.

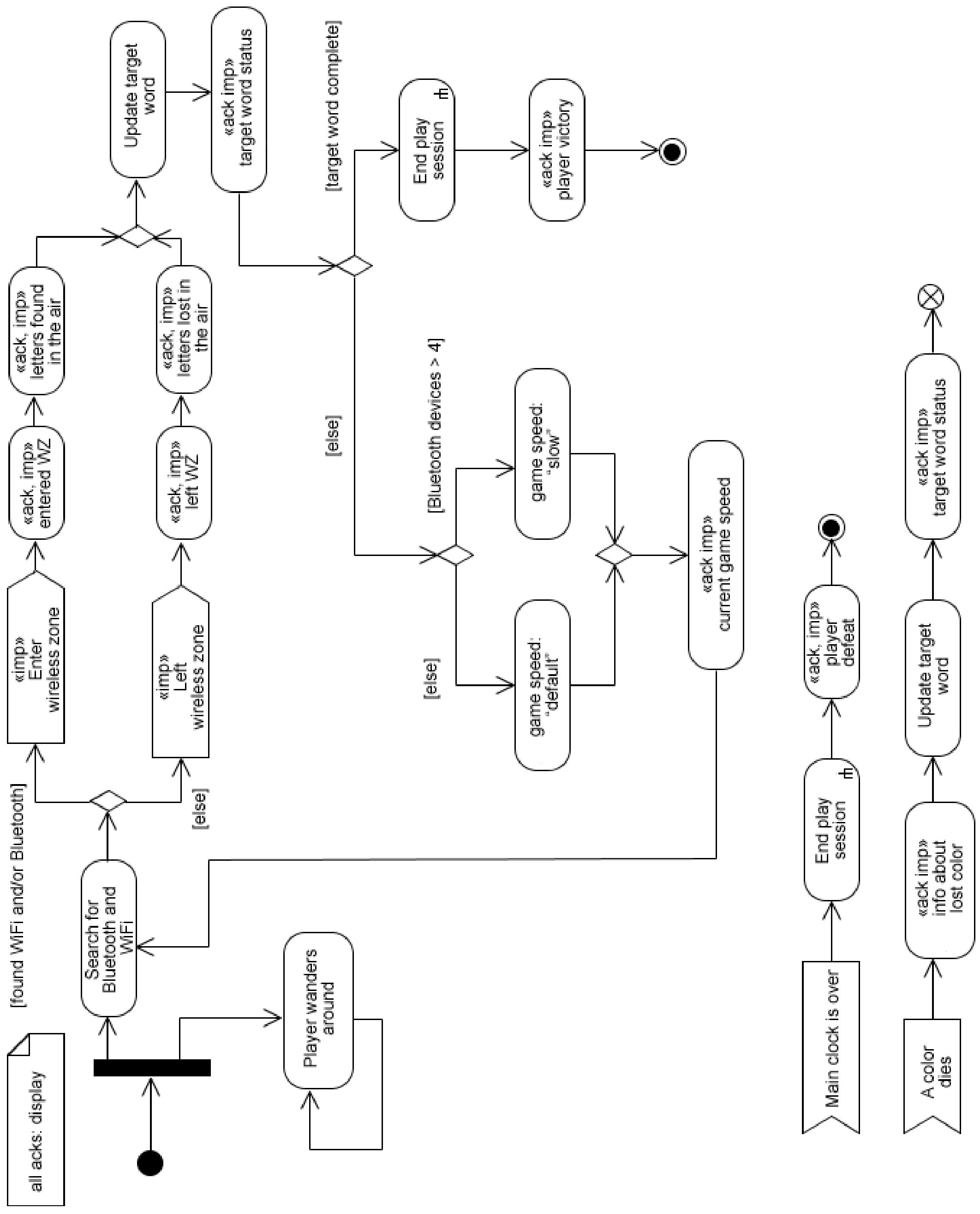

Figure 8.5: *Interact with wireless zone* activity diagram

The events *Main game clock runs out* and *A color dies* are the same that appeared the in *Capture color* activity. In this game, they are events that may happen in all game sessions, possibly affecting all activities (*e.g.* global scope). Figure 8.6 illustrates the textual specification of this activity.

Scope	Primary
Primary Actors	Player
Sensors	Bluetooth, WiFi
Actuators	Display
Interaction granularity	Continuous
Control	Implicit

Notices
This activity loops while there is an active play session.

Default flow
1. Player wanders around in the physical environment
2. Game finds Bluetooth and WiFi devices, updates current wireless set
   - **ack (display)**: entered wireless zone
   - **ack (display)**: new and lost letters in the current wireless set
3. Game updates target word with letters from the current wireless set
   - **ack (display)**: target word status
4. Game adjust time speed to "default"
   - **ack (display)**: time speed status

Alternative flows
2a. Number of Bluetooth and WiFi devices is zero:
1. Notifications
   - **ack (display)**: left wireless zone
   - **ack (display)**: all wireless letters are lost
2. Game updates target word state
   - **ack (display)**: target word status
3. Game restarts activity

3a. All letters of target word have been captured:
1. Perform ***End play session***
   - **ack (display)**: announces player victory
2. **EOA**

4a. Number of Bluetooth devices is greater or equal than 4:
1. Game adjusts time speed to "slow"
   - **ack (display)**: time speed status

4b. Number of Bluetooth devices is less than 4:
1. Game adjusts time speed to "default"
   - **ack (display)**: time speed status

*a. Main game clock runs out:
1. Perform ***End play session***
   - **ack (display)**: announces player defeat
2. **EOA**

*a. A color dies:
1. Update target word state
   - **ack (display)**: info about lost color
   - **ack (display)**: current target word status

Uncertainty handling policy
- Hide
  - Announce 'enter zone' and 'leave zone' events only – do not inform query status (started, finished, in progress, identify devices found and lost, etc.).
  - Do not provide area maps with devices

Figure 8.6: Text specification of "*Interact with wireless zone*"

The interaction granularity is *continuous*, as the process of querying devices and the results of this process are not predictable. This activity has implicit style as the player has no direct control over it.

The game performs this activity continuously while there is a play session happening. In case of alternative flow `2a`, the third step states that the activity should restart at that point instead of going back to the default flow (which would not make sense if flow `2a` is invoked).

There are some steps that have more than one acknowledgment (*e.g.* step `2` in the default flow). In all activities for this game, the acknowledgments do not have to follow a specific sequence, so they are indicated as unnumbered lists. If they should be delivered in a specific order, they should be specified using numbered lists to reflect the sequence.

This activity starts automatically when the play session starts – hence it is of *implicit* style, from the player point of view. The granularity of interaction is *continuous*, as the process of querying devices and its results are not predictable.

The *Uncertainty handling policy* is *hide*. This activity realizes this strategy by presenting information to the player in imprecise and ambiguous ways ("ambiguity of representation"). For example, Bluetooth and WiFi queries are slow operations. In our experiments, a Bluetooth query could take up to 10s to complete. It is also possible that some queries miss some devices, or return false positives. Hence, this activity does not require using Bluetooth data in real-time fashion.

The representation of the zones in the game is deliberately imprecise and ambiguous – it does not display a map of the zones, nor the events related to the queries. Instead, the game just informs if the player has entered or left a zone[94].

8.1.3 Interact with open zone

In *Pervasive Word Search*, an open zone corresponds to an outdoor area. This activity takes place when the game detects that the player is outdoors.

94 This tactic also applies to other game zones in the game (dark and open zones), controlled through other activities.

When the player is in an open zone, the game clock runs faster, affecting the life span of colors (which "die" faster). The game alerts the player to go to a "safe place" (an indoor area), where the game clock turns back to normal behavior.

This is an *implicit-styled* activity – the player does not interact with the open zone through direct commands. Instead, the game scans for an open zone in the background, while the player wanders around.

Figure 8.7 illustrates the diagram of this activity.

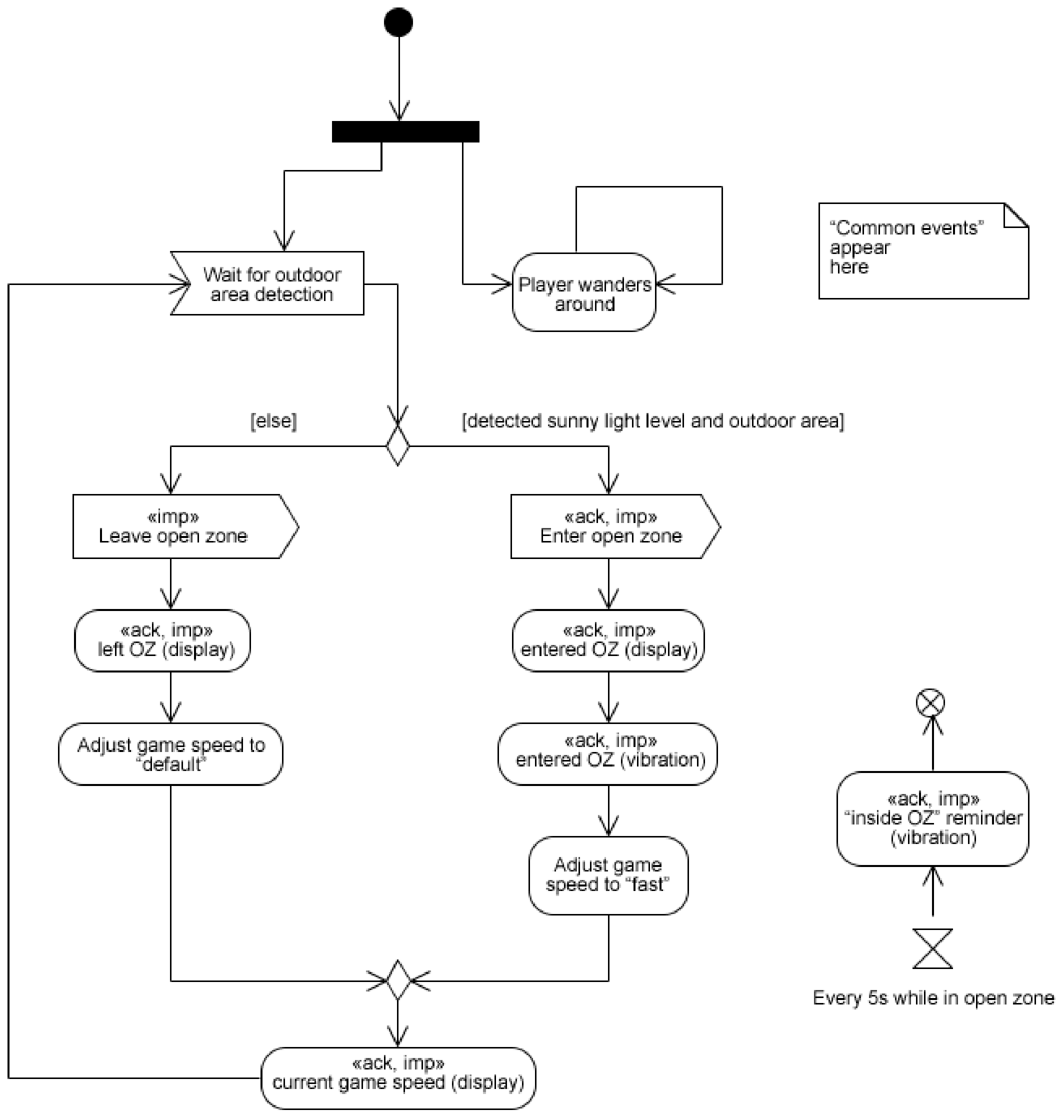
Wait for outdoor area detection
Player wanders around
"Common events" appear here
[else]
[detected sunny light level and outdoor area]
«imp» Leave open zone
«ack, imp» Enter open zone
«ack, imp» left OZ (display)
«ack, imp» entered OZ (display)
Adjust game speed to "default"
«ack, imp» entered OZ (vibration)
«ack, imp» "inside OZ" reminder (vibration)
Adjust game speed to "fast"
Every 5s while in open zone
«ack, imp» current game speed (display)

Figure 8.7: "*Interact with open zone*" activity diagram

This example illustrates using the following concepts:

- Activity with implicit style, and continuous granularity;

- Acknowledgments (implicit style);
- Invoking other activities;
- Branching and merging flows;
- Concurrent flows;
- Generic, interaction, and time events;
- Using notes to suppress repeated alternative flows.

In Figure 8.7, we used a note labeled with “common events appear here” to avoid repeating the generic events *Main game clock runs out* and *A color dies* in this diagram. In this case, we create a separate diagram to host these two events, and reference this diagram when necessary.

This activity starts when the play session starts, without player intervention. While the player walks around, the activity waits for an outdoor area detection response. When this event occurs (*e.g.* the device sensors respond about the current outdoor condition), the activity may affect the game clock behavior if the player is outdoors – the interaction event “Enter open zone” occurs, and the game produces two acknowledgments “entered open zone” (using display and vibration motors). While inside an open zone, the game alerts the player about it, by delivering acknowledgments periodically through vibration as consequences of a time event. If the player leaves the open zone, the game generates the interaction event “Leave open zone”, delivering the acknowledgment “left open zone”, and the game clock behavior comes back to normal.

Figure 8.8 illustrates the text specification for this activity. The interaction granularity is *continuous*, and the style is *implicit* as the player has no direct control over it. The activity starts when the play session starts.

The *Uncertainty handling policy* is *hide*. This activity realizes this strategy by presenting information to the player in imprecise ways (“ambiguity of representation”). For example, the game only announces “enter” and “leave” events, not providing much detail about how an open zone is defined.

```
Scope                           Primary
Primary Actors                  Player
Sensors                         light sensor, GPS
Actuators                       Display, vibration motor
Interaction granularity         Continuous
Control                         Implicit

Notices
This activity loops while there is an active play session.

Default flow
  1. Player wanders around in the physical environment

  2. Game detects an outdoor area with light sensors and GPS
     • ack (display): entered open zone
     • ack (display): entered open zone

  3. Game adjusts clock speed to "fast"
     • ack (display): new clock speed status

  4. While in open zone, Game reminds Player about being in an open zone every 5s
     • ack (display): "inside open zone" reminder

Alternative flows
2a. Outdoor area not detected:
  1. Notifications
     • ack (display): left open zone

  2. Game adjusts clock speed to "normal"
     • ack (display): target word status

  3. Game restarts activity

*a. "Common generic events" appear here

Uncertainty handling policy
  • Hide
     ▫ Announce 'enter' and 'leave' events only
     ▫ Use a very bright light level to infer an "outdoor area", along with GPS
```

Figure 8.8: Text specification of "*Interact with open zone*"

This example avoids repeating the common events by indicating them as "`*a. "Common generic events" appear here`". In this case, these common events should be specified separately in the document, as Figure 8.9 illustrates.

```
Common generic events

Overview
These are events that may affect all activities in this game, working as altern-
ative flows for all of them. Separated here to avoid repetition.

Alternative flows
*a. Main game clock runs out:
  1. Perform End play session
     • ack (display): announces player defeat
  2. EOA

*a. A color dies:
  1. Update target word state
     • ack (display): info about lost color
     • ack (display): current target word status
```

Figure 8.9: Isolating common events in their own section

8.1.4 Interact with dark zone

A dark zone corresponds to an area with “low-light” conditions[95]. When a player enters a dark zone the game grants the player white and gray colors, which receive infinite time span while the player is in this zone. If the player leaves this zone, he loses white and gray colors. As the other zone interactions, interacting with the dark zone has implicit style and this starts along with the play session. Figure 8.10 illustrates the diagram for this activity.

This example illustrates using the following concepts:

- Activity with implicit style, and continuous granularity;
- Acknowledgments (implicit style);
- Interaction events;
- Invoking other activities;
- Branching and merging flows;
- Concurrent flows;
- Using notes to suppress repeated alternative flows and to indicate common actuators in use for acknowledgments.

This example also uses the note labeled with “common events appear here” to avoid repeating generic events. Also, all acknowledgments are delivered through the device display, hence the note indicating this. We used these measures to make the diagram cleaner.

The interaction granularity is *continuous*, and the style is *implicit* as the player has no direct control over it. The *Uncertainty handling policy* is *hide*. This activity realizes this strategy by announcing “enter zone” and “leave zone” events only, as the game does with the other zone interactions.

95 We will consider as “dark” the lowest light level detectable by the device. However, in practice the results depend on the device hardware.

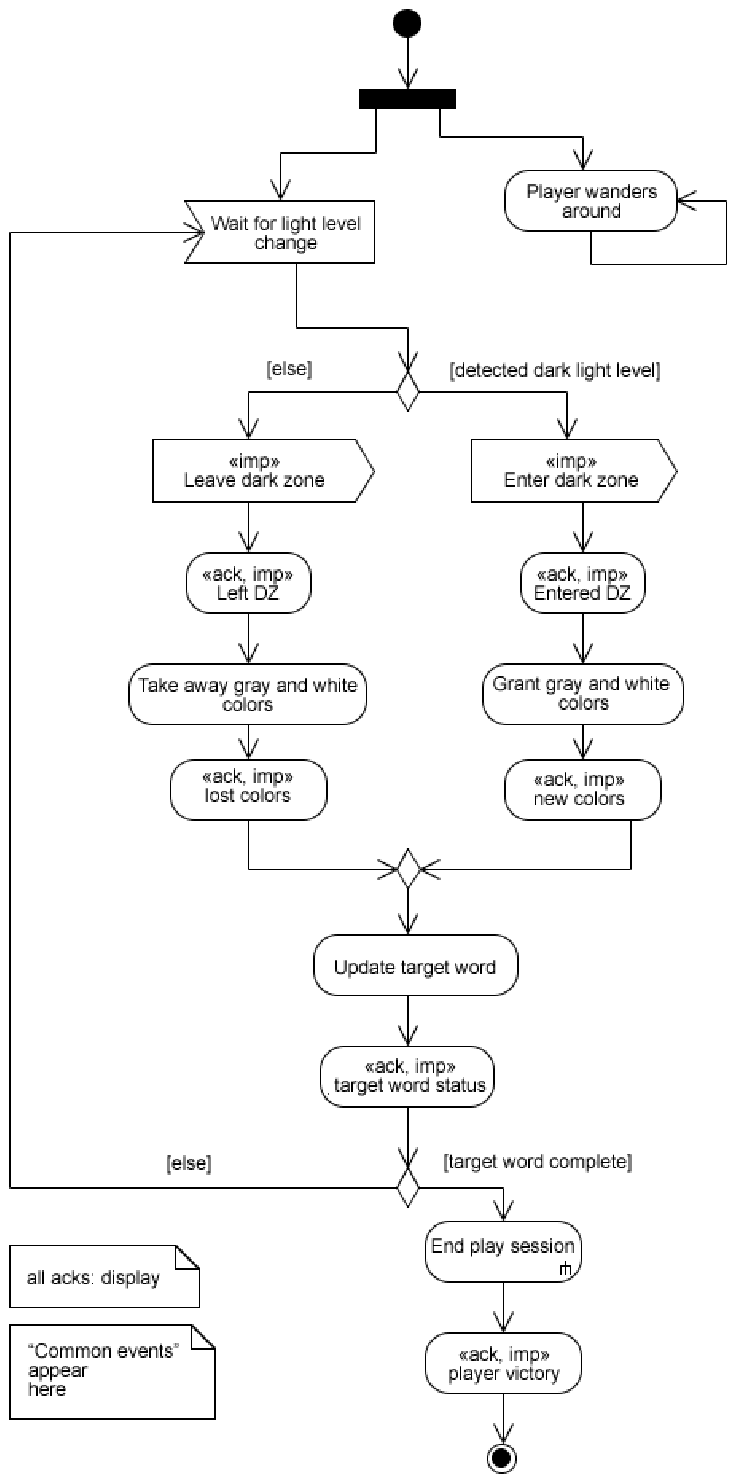

Figure 8.10: "*Interact with dark zone*" activity diagram

Figure 8.11 illustrates the text specification for this activity.

```
Scope                          Primary
Primary Actors                 Player
Sensors                        light sensor
Actuators                      Display
Interaction granularity        Continuous
Control                        Implicit

Notices
This activity loops while there is an active play session.

Default flow
  1. Player wanders around in the physical environment

  2. Game detects a "dark zone"
     • ack (display): entered dark zone

  3. Game grants player bonus colors
     • ack (display): the granted colors
     • ack (display): target word status

  4. Game restarts activity

Alternative flows
2a. Dark zone not detected:
  1. Notifications
     • ack (display): left dark zone

  2. 2. Game takes away bonus colors from player
     • ack (display): the lost colors
     • ack (display): target word status

  3. Game restarts activity

3a. Target word complete:
  1. Perform End play session
     • ack (display): declare player as winner

*a. "Common generic events" appear here

Uncertainty handling policy
  • Hide
     ▫ Announce 'enter' and 'leave' events only
```

Figure 8.11: Text specification of "*Interact with dark zone*"

8.1.5 End play session

This is a secondary activity that all primary activities reference. Figure 8.12 illustrates the diagram for this activity.

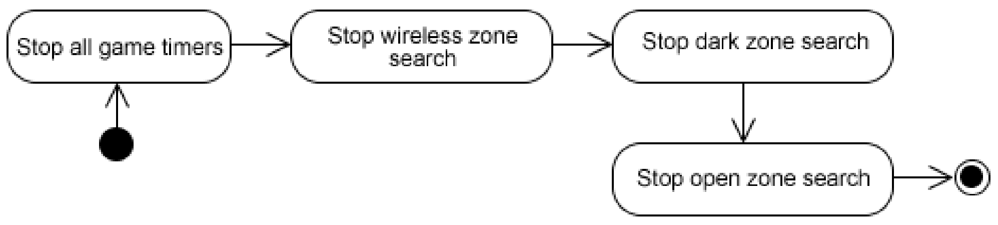

Figure 8.12: "*End play session*" activity diagram

As several activities need to end the play session at some time, we opted to separate this specific behavior into an activity that can be reused. Figure 8.12 illustrates the textual specification for *End play session*.

```
Scope                         Secondary
Primary Actors                Player
Sensors                       All
Actuators                     -
Interaction granularity       -
Control                       -

Overview
All concurrent activities are interrupted and shut down.

Default flow
  1. Stop wireless zone search (Bluetooth, WiFi)
  2. Stop dark zone search (light sensors)
  3. Stop open zone search (light sensors, GPS)
```

Figure 8.13: Text specification for "*End play session*"

8.2 Other examples

In this section we provide an additional example of using the DSL for specifying the *Search for treasure in the room* activity in *The Audio Flashlight 5*[96] prototype, a game for three players. In this game two players (Flashlight 1 and 2) have to find a hidden treasure in a dark room before the game clock runs out, and the third player (Spoiler) is responsible for disturbing the others (by moving the treasure) and hoping that the others cannot find the treasure before the clock runs out. This example illustrates using the following concepts:

- Activity with explicit style, and discrete granularity
- Distributed activity
- Acknowledgments
- Interaction events
- Invoking other activities
- Branching flows
- Concurrent flows (fork and join)
- Comments in diagrams

Figure 8.14 illustrates the diagram of *Search for treasure in the room*.

96 Please refer to Appendix C.2 for details about this prototype.

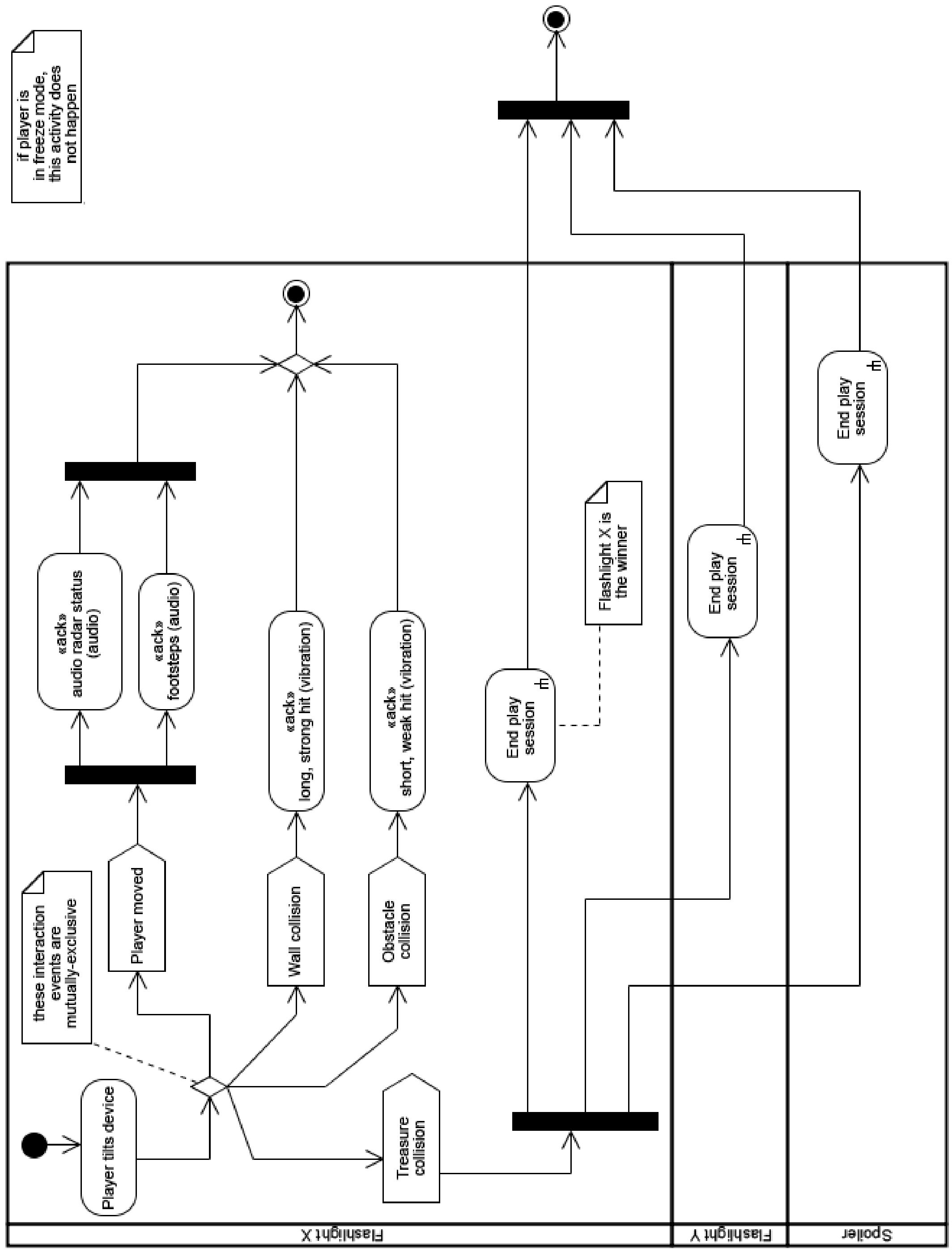
if player is in freeze mode, this activity does not happen
Player tilts device
these interaction events are mutually-exclusive
Player moved
Wall collision
Obstacle collision
Treasure collision
«ack» audio radar status (audio)
«ack» footsteps (audio)
«ack» long, strong hit (vibration)
«ack» short, weak hit (vibration)
End play session
Flashlight X is the winner
End play session
End play session
Flashlight X
Flashlight Y
Spoiler

Figure 8.14: Diagram for "*Search for treasure in the room*"

The Flashlight players move in the virtual world using gestures, and have to hear the audio and sense the vibrations to navigate in the virtual world. The Spoil-er player does not move.

This activity is performed by Flashlight 1 and 2. The Spoiler player may interfere in this activity by starting *Move treasure to another place in the room*, when he changes the treasure position.

The activity starts when the player tilts the mobile phone. The default flow happens when the player successfully moves in the game room (the interaction event "Player moved"). The alternative flows are represented with the other interaction events coming from "player tilts device". Decision nodes usually require guards to specify the conditions for the branching. In this case, adding the guards would only clutter the diagram as the interaction events are mutually-exclusive. To make this clear, the diagram includes a comment (a note) informing this.

When the Flashlight player finds the treasure (represented as the interaction event "Treasure collision"), it invokes the secondary activity *End play session*. Essentially, this activity notifies the other players about the winner and losers, and ends the level. If there are more levels, the game loads the next one. Otherwise, the game presents the ending to the players.

The *Uncertainty handling policy* is *hide*. The accelerometer is an analog sensor, producing continuous signals. A solution to implement this is to normalize the produced output range, using "known" (and "well-defined") positions for the walk gestures, and providing the appropriate acknowledgments. Figure 8.15 illustrates the gestures.

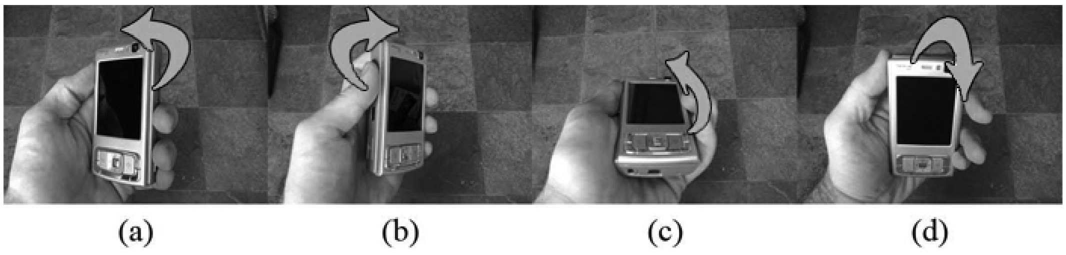

Figure 8.15: Basic moves. Go left (a), right (b), forward (c), and backward (d)

In this sense, the game uses an analog (continuous) sensor for a discrete-styled interaction. In the research on the original *The Audio Flashlight* game (Valente *et al.* 2008; Valente *et al.* 2009), players have reported enjoying this kind of interaction.

Figure 8.16 illustrates the text specification of this activity.

Scope	Primary
Primary Actors	Flashlight X
Secondary Actors	The other Flashlight player (Y), Spoiler
Sensors	Accelerometer
Actuators	Vibration motor, speakers
Interaction granularity	Discrete
Control	Explicit

Overview
Player tilts device to move in the virtual environment, guided by the audio radar, until he finds the treasure. There is a time limit to find the treasure.

Default flow
1. Flashlight X interacts with the device, positioning it into four possible positions for walking

2. Game moves Flashlight X's avatar in the room
   - **ack (audio):** new audio radar status
   - **ack (audio):** footsteps

Alternative flow
1a. Game is in Freeze Mode:
- Perform ***Interact in Freeze Mode***

1a. Flashlight X chooses the "idle position" (no move)
- **ack (audio):** stop footstep audio

1b. Flashlight X chooses the "end game" position
- Perform ***Quit Game***

2a. Game does not move Flashlight X, there's an obstacle in the way:
- **ack (vibration):** weak and fast (intensity: 20%, duration: 40ms)

2b. Game does not move Flashlight X, there's a wall in the way:
- **ack (vibration):** strong and long (intensity: 100%, duration: 500ms)

2c. Game moves Flashlight X's avatar and he finds the treasure:
- Flashlight X won; Perform ***End play session***
- **EOA**

*a. Level stopwatch reaches zero:
1. Spoiler won; Perform ***End play session***
2. **EOA**

Uncertainty handling policy
- Hide: Normalize accelerometer range, and use known phone positions (discrete) for the walk gestures

Figure 8.16: Text specification of "*Search for treasure in the room*"

The default flow is a Flashlight player interacting with the mobile phone and walking in the room. The alternative flows describe cases where the player is unable to move, and how other players (the other Flashlight player and the Spoiler) are affected when the Flashlight player finds the treasure. There is also an alternative flow (`1a`) for the case where the players are "frozen"[97]. In this case, the activity *Interact in Freeze Mode* takes place.

The "freeze mode" is invoked by a Flashlight player and affects the other Flashlight player and the Spoiler player. When in "frozen state", their devices start

97 Please refer to Appendix C.2.5 for details about this prototype and the "freeze mode".

malfunctioning, which means that the Flashlight player is unable to move and the Spoiler player is unable to change the treasure location and to see the game map.

8.3 How diagrams were useful

Using diagrams helped prototype development in several ways. This section provides a discussion on this subject.

8.3.1 Finding missing flows in textual specification

In *Pervasive Word Search*, the activity *Interact with wireless zone* (Section 8.1.2) was first specified as text. The diagram for this activity was devised later as a way to review the textual specification (Figure 8.5).

When drawing the diagram, it became clearer that the specification was missing an alternative flow (`2a`) and the default flow was missing a step (`2`). The textual specification was updated to include the missing flow and missing step.

8.3.2 Finding errors in code

As a short example, consider the activity *Search for treasure in the room* from *The Audio Flashlight 5* (and previous versions). When a player moves, the game should deliver two acknowledgments concurrently (Figure 8.14), as audio (footsteps and audio radar status). In this specific example, the actions of delivering each acknowledgment do not interfere with each other.

After drawing the diagram for this activity, and using it for a review of the implementation, we were able to catch inconsistencies in the code.

In the original implementation, the method for updating the player position (after the game recognized the command for walking) was doing more than it should do: it was also providing acknowledgments. Figure 8.17 illustrates a simplified excerpt of the affected source code.

```
char updatePlayer()
{
    // player idle?
    if (_player.currentDirection() == Player::MOVE_STOP)
      { stopFootSteps(); return GameMapElements::FREE_CELL; }


    char hit = _player.move(_gameMap);
    generateFeedback (hit);

    return hit;
}

void updatePlayingState ()
{
   char hit = updatePlayer();

   if (hit == GameMapElements::TARGET)
   {
      endPlaySession();
      return;
   }

   updateSound();

}
```

Figure 8.17: Code excerpt for updating the player state and acknowledgments

The `updatePlayingState()` method is responsible for updating the main playing state for the game. It should update the player position (according to previously received commands), and generate the appropriate acknowledgments. The `updateSound()` method updates the audio radar status according to the player position. The `updatePlayer()` method tries to move the player, returning information about collisions (walls, obstacles, target, or no collision). The `generateFeedback()` method was responsible providing appropriate acknowledgments according to the collision information from the `updatePlayer()` method (footsteps, or vibrations in case of collision).

A consequence of this implementation is that `updatePlayer()` was also responsible for providing the acknowledgments. In this regard, swapping the calls of `updatePlayer()` and `updateSound()` would produce side effects – the audio radar status would be inconsistent for at least one execution cycle of the main game loop. The side effects should not appear as those acknowledgments should occur concurrently (and independently), according to the activity specification.

The solution is to make generating acknowledgments as part of `updatePlaying()`, and not from `updatePlayer()`.

Although this is a simple example, the point is that the diagrams can be a useful tool to check for implementation mistakes.

8.3.3 Keeping focus on intent

Keeping focus on intent may be the most important utility for using diagrams. The UML Activity Diagrams have a level of abstraction that is enough to specify activity flow on a higher level (intent). This has helped to guide the implementation without getting lost in low level programming details.

For example, the prototypes were implemented using C++ and the Qt framework[98] (Nokia 2011; Blanchette and Summerfield 2008). The architecture of the Qt framework is greatly based on the concepts of *signals and slots* (Qt 2011). Signals and slots are language constructs in Qt for implementing communication between objects. In practice, it works like the Observer design pattern (Gamma *et al.* 1995). A signal is emitted when a specific event occurs. A slot is a method of a class interested in receiving notifications of signals. Qt provides C++ language extensions to declare signals, slots, and to connect them. When a signal occurs, all connected slots are called.

In this regard, a Qt application often has many parts that are asynchronous. More specifically, the Qt APIs for handling sensors (part of the Qt Mobility package[99]) are heavily based on signals and slots.

As a consequence, the code to implement activities has become scattered. In this sense, the diagrams have served as a high-level roadmap to the activity flow, helping to cope with the code scattering.

Also, sometimes when implementing the activities, new ideas have emerged. For example, new acknowledgments, variation of game rules (affecting the activity implementation). As the diagrams do not have a high level of (implementation) detail, it became easier to update the diagrams with the new ideas.

98 Except for *Location-based Quiz Game*, which has been developed with Java.
99 For more information on those APIs, please refer to (Qt Mobility 2011).

8.4 Summary

This chapter presented the final part of our case study – *Pervasive Word Search*. This final part corresponds to applying the DSL to specify all activities for this game, both in text and visual formats. A total of five activities have been exemplified.

This chapter also presented the activity *Search for treasure in the room* from *The Audio Flashlight 5*, which made it possible to demonstrate the concepts in the context of a multi-player game. The chapter ends with discussing how using diagrams was useful for activity specification and implementation.