Educational Virtual Game Scenario Generation for Serious Games

Olivier Janssens*, Koen Samyn†, Rik Van de Walle ‡ and Sofie Van Hoecke*‡

*Electronics and Information Technology lab, ISP, Ghent University
Graaf Karel de Goedelaan 5, 8500 Kortrijk, Belgium,
Telephone: + 32 56 24 12 52; email: odjansse.janssens@ugent.be

†Digital Arts and Entertainment, University College West-Flanders

‡Multimedia Lab, ELIS, Ghent University - iMinds

***Abstract*—In order to help youngsters who are confronted with cyberbullying, the Friendly ATTAC project aims to develop a serious game. In this paper the novel virtual game scenario generation process for this serious game is presented. The goal of the process is to allow non-technical users to model virtual scenarios. After writing the scenarios, the scenarios can be modelled in ATTAC-L and afterwards translated in computer interpretable XML. This XML is then used to automatically build the scenario within the game engine so that it can be played. This paper details the ATTAC-L scenario generation and how it is translated to an in game scenario. Thanks to the presented tools and methods, it is possible to transform a written scenario into an in game virtual game scenario.**

I. INTRODUCTION

Cyberbullying (bullying via electronic communication tools) is a relatively recent phenomenon that especially occurs among early adolescents (12-15 year olds). As cyberbullying may have a serious impact on the mental (and physical) well-being of victims, many societal actors are currently involved in anti-cyberbullying initiatives. They strongly plea for evidence-based, appealing, ICT-related intervention tools that empower youngsters confronted with cyberbullying.

The Friendly ATTAC project [1] studies and develops an innovative serious game to help youngsters deal with cyber-bullying issues. By means of highly personalized virtual experience scenarios, providing players with immediate feedback in a safe computer-mediated environment, the project attempts to modify behavior patterns of bullies, bystanders and victims. This is done by allowing youngsters, through the use of the virtual scenarios, to experience different roles (bully, victim, or bystander) in cyber bullying incidents during the game, to react to those experiences, and to get adjusted feedback based on their individual reactions. This will increase their empathy, enhance their social skills or teach/train them relevant coping strategies.

The scenarios and interactions within the serious game are based on theoretical and empirical knowledge regarding personal and contextual determinants of cyber bullying that are obtained by performing a well-established intervention method from the field of health psychology: Intervention Mapping (IM) [2].

One of the challenges of the project is to translate these theoretical and empirical knowledge regarding personal and contextual determinants and causes of cyber bullying into attractive virtual scenarios for youngsters. As the scenarios are developed by social scientists, health psychologists, computer scientists, people working in the field, and game designers, the interdisciplinary collaboration requires tools and methods that can be easily used and understood by all parties involved, also non-technical users. These tools can be used to develop new scenarios and game levels and semi-automatically create the actual game. This is not only important in the context of the project, but also later on (after the end of the project) when new scenarios need to be developed for the same or a related domain.

To achieve this, a domain-specific modeling language is developed to accommodate domain experts in the design process of these scenarios/games. They can use this domain-specific modeling language, called ATTAC-L, to write down their game scenarios and have it automatically translated to an XML file, which can be used in the code generator to translate the scenario into actual game moves.

The remainder of this paper is as follows. The next section outlines serious games and the general concept of our solution to semi-automatically translate theoretical and empirical knowledge into attractive virtual scenarios. Subsequently, in Section III we define the features of the domain-specific modeling language, ATTAC-L, and in Section IV how it is translated into an XML file. Section V covers the scenario code generator that translates the scenario into actual game moves. Next, Section VI describes the resulting game, while Section VII covers the testing framework. Finally, in Section VIII we summarize the most important conclusions of our work and our plans for the future.

II. GENERAL CONCEPT

Serious games are computer or video games that are not only designed for fun, but also have an educational purpose. It is however essential and challenging to maintain the ballance between the game element and the learning aspect, otherwise the desired effect (e.g. impact on health behavior, enhancing social skills) is missed.

The choice for a tailored serious game to promote health to youngsters is certainly supported by scientific evidence [3]. Tailoring, also called personalised advice, refers to creating custom health messages for each person to make these messages more relevant. For each learning outcome, different game features lead to different effects. For example, for teaching skills and increasing self-efficacy, simple games without storyline are better suited. On the other hand, for teaching knowledge and attitudes, both simple and complex games can be used. This implies that a game aiming for different learning outcomes should be best divided into several subgames so the right style can be chosen for the learning objective.

For the Friendly ATTAC serious game, focussing on helping youngsters dealing with cyberbullying issues, the school context seems feasible. This way, on one hand, the likelihood increases that also poorly motivated youngsters are reached to adapt their social behavior, and on the other hand, a pretty simple game can be developed to draw their attention. Moreover, the game can be plugged into the curriculum as the involvement of schools is a key factor in addressing bullying.

Scenario/level generation is often done via procedural content generation (PCG), which refers to creating game content automatically, through algorithmic means [4],[5]. In this research several steps have to be followed in order to generate a scenario for the serious game. The steps can be seen seen in Figure 1. Firstly, social scientists have to write a scenario which incorporates their theories and hypothesis they want to test or model. Secondly, via an authoring tool they can translate their written scenario into an ATTAC-L scenario. The ATTAC-L language is a graphically illustrated modeling language for educational virtual scenarios, understandable by non-technical people. The third step consists of the XML file generation based on the ATTAC-L scenario. By using XML, it is possible for a computer to read and interpret the scenario. The fourth step takes the XML file and translates it into actual game moves.

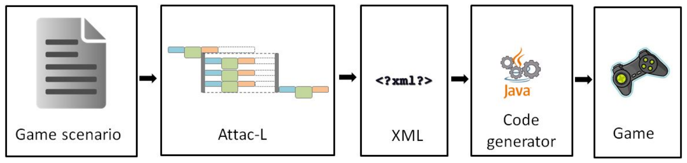


Fig. 1: General concept of process to generate game scenarios

The different tools and methods have been implemented and are currently used to build the different levels of the serious game for the Friendly ATTAC project. In the next sections, the different steps of the process are described in more detail. In order to illustrate the entire process, a use case is provided throughout this paper.

Use case

Setting: There are three people namely the player, *John Smith* (non-playable character) and *Kate Johnson* (non-playable character). Kate is 16 years old and female. John is 15 years old and male. They all go to school together and all know each other, but only the player and John are friends and respect each other. Both the player and John do not like Kate since she is a bully, and therefore they do not respect her.

Situation: Kate sends a tweet to John saying he better not show his face tomorrow at school or bad things will happen. Luckily the player is brave and tells Kate to shut up and John to ignore the tweets. Kate is angry and warns them both to watch out tomorrow because bad things will happen. Now the player gets a choice to ignore the tweet or respond with the text “Bring it on”. Depending on the chosen response Kate won’t tweet anything or tweet “HAHAHA”.

III. ATTAC-L

Rather than resorting to natural language to specify the scenarios, a more formal approach is used, which has the advantage over natural language of not being ambiguous and allowing to a certain extent the semi-automatically creation of the actual games.

To achieve this, a domain-specific modeling language, called ATTAC-L is developed to accommodate domain experts in the design process of these scenarios/games. A domain-specific modeling language [6] is a language that uses a dedicated vocabulary and provides abstractions that make the specifications of solutions easier and more accessible for domain experts. The domain-specific modeling language is implemented as a graphical language as these graphical specifications are easier for the communication with non-technical people. These graphical or visual specifications are easier for the communication with non-technical people than textual languages; they are also helpful for conveying complex models and designs as they can help people to grasp large amounts of information more quickly than large listings of text.

The general idea of ATTAC-L is to express the story of a game in a way as intuitive as possible. Therefore, a combination of flowcharts and natural language like syntax is used. The natural language like syntax is used to specify the individual game moves in the game, while the flowchart approach is used to express the chronological order between the game moves. By a natural language like syntax, we mean that sentences are used that look/read like simple natural language sentences but actually have a strict syntax to make them understandable by the computer. The sentences are also not expressed as text but by means of a graphical notation using bricks.

Bricks are the basic building blocks for specifying game moves, i.e. the actual steps that will be performed in a game. The bricks are organized in different categories, indicated by mean of a color, and can be connected to each other according to certain rules that are influenced by the grammatical rules from the natural language syntax. The result is a construct that unambiguously describes a game move, expressed in a human-readable form.

Creating an Non-Playable Character (NPC)

Besides using ATTAC-L for specifying the player’s game moves, it is important to have other characters apart from the player as well. ATTAC-L can also be used to create these Non-Playable Characters (NPCs).

For the presented use case, two NPCs are created, i.e. Kate and John. The NPC character creation using ATTAC-L for the presented use case can be seen in Figure 2).

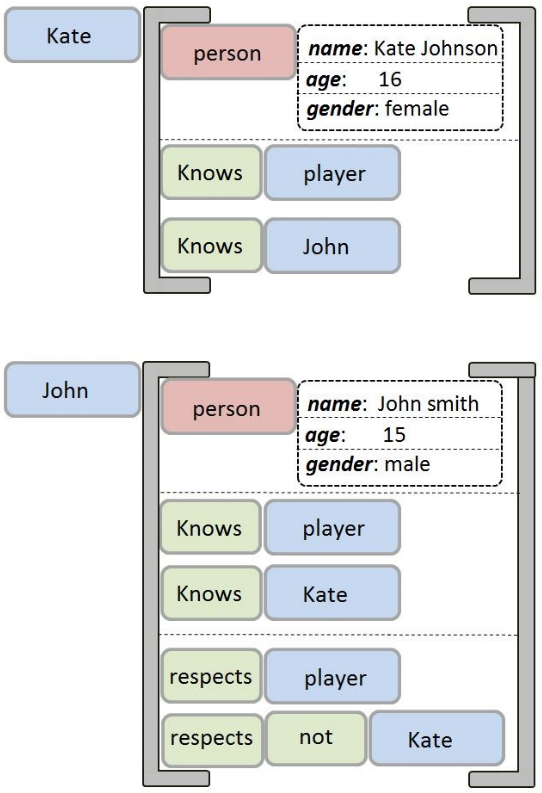


Fig. 2: Non-playable character creation via ATTAC-L

As can be seen in this figure, both Kate and John get defined as a person with certain properties. Also the relationships between the NPCs are defined in this declaration.

Communication

As the Friendly ATTAC serious game is a game against cyberbullying, it is important to have communication between the characters in the game. An example of the player sending a tweet to NPC Kate can be found in Figure 3.

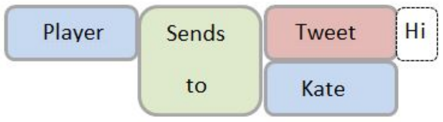


Fig. 3: ATTAC-L example of sending a tweet

Sequence

A *sequence* is used to indicate a sequential order between game moves and is graphically represented by a *sequence* brick in ATTAC-L. A sequence brick is represented as an empty gray block connecting the ending and starting brick of the two consecutive game moves. Figure 4 presents the player sending a tweet to Kate, who subsequently tweets back to the player.

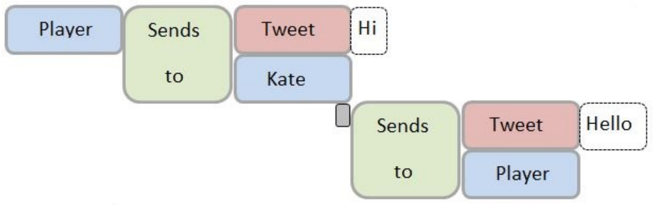


Fig. 4: Example of a sequence in ATTAC-L

Choice

A *choice* defines a point in the scenario where this scenario is split into two or more alternative paths and a decision has to be made on which path to follow. To indicate the choice, a *choice* brick is used in ATTAC-L. Figure 5 gives the player the choice between two tweets to be send to Kate.

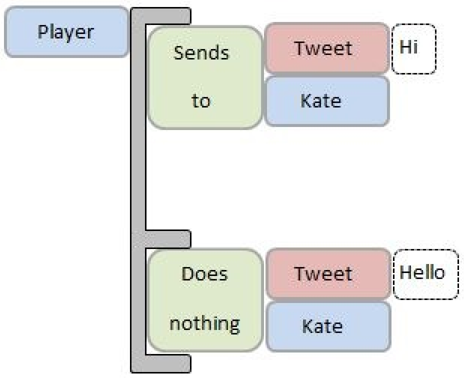


Fig. 5: Example of a choice in ATTAC-L

Concurrency

Apart from a *choice*, *concurrency* is also possible. A *concurrency* structure encapsulates two or more paths that are followed concurrently at the same time. The different paths are started together but are not required to end together. An example can be seen in Figure 6 where the player concurrently tweets Hi and Hello to Kate.

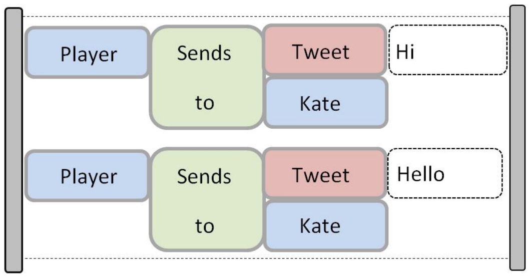


Fig. 6: ATTAC-L example of concurrency

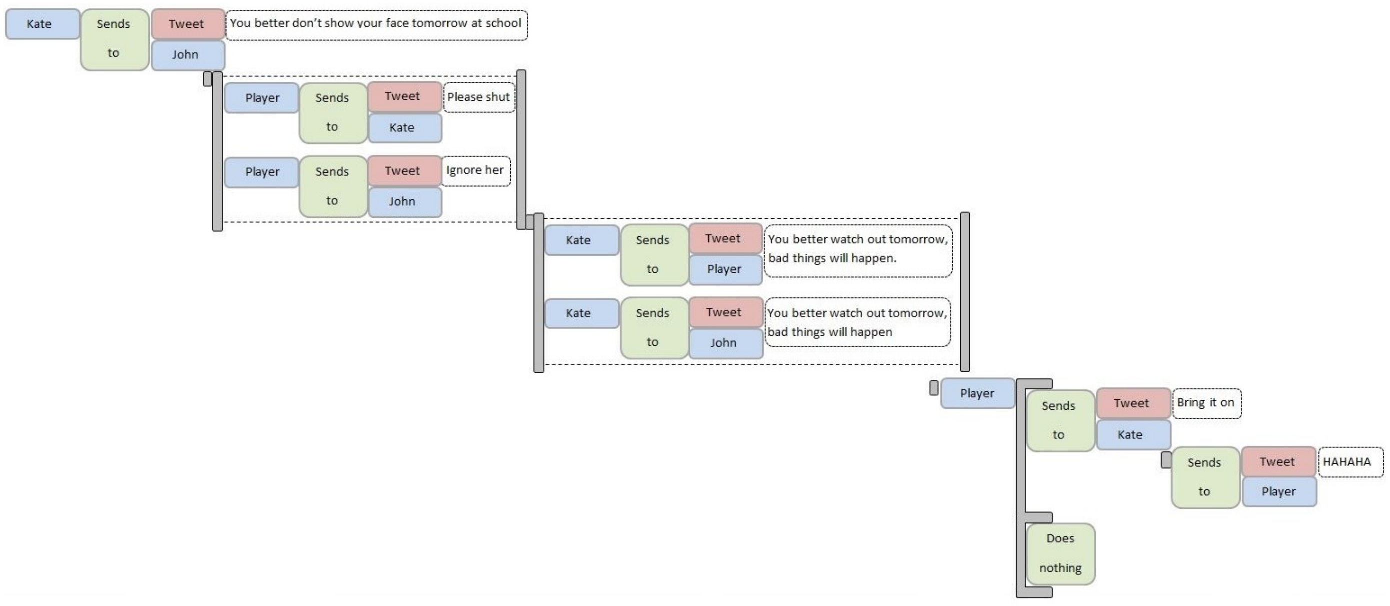


Fig. 7: Use case translated to ATTAC-L

Use case

By combining the above ATTAC-L structures, it is possible to translate the complete use case to ATTAC-L. The declaration of the NPCs in ATTAC-L is not repeated here as this can be found in Figure 2; the rest of the use case scenario can be found in Figure 7.

For more information on the ATTAC-L domain specific modeling language, the reader is referred to [7].

IV. ATTAC-L TO XML CONVERSION

Once a scenario is constructed in ATTAC-L using the authoring tool, this scenario can be translated to an XML file, using the same authoring tool. The authoring tool is web-based and uses HTML, JS and CSS to let users easily construct the scenarios. In the back-end, these scenarios are translated to XML so they can be used by the code generator to translate them into actual game moves.

In this section, the XML structures are given for the matching ATTAC-L examples of the previous section.

Creating an Non-playable character (NPC)

The ATTAC-L creation of Kate and John, the NPCs in the game, is translated into the following XML:

Listing 1: Non-playable character creation via ATTAC-L

```
<defineObject type="person" name="Kate" actorid="1"
sex="female" age="16">
 <verb type="know">
  <who id="2" negate="false"/>
  <who id="0" negate="false"/>
 </verb>
</defineObject>

<defineObject type="person" name="John" actorid="2"
sex="male" age="15">
 <verb type="know">
  <who id="1" negate="false"/>
  <who id="0" negate="false"/>
 </verb>
 <verb type="respect">
  <who id="1" negate="true"/>
  <who id="0" negate="false"/>
 </verb>
</defineObject>
```

As in ATTAC-L, the XML structure has specific tags for certain bricks such as the *verb* tag or the *who* tag. It is important to note that both NPCs and the player get an id. These ids allow to specify the involved characters for each action. Eventually when this piece of XML is interpreted by the code generator, an NPC is spawn in the game which has connections to other characters.

Communication

The ATTAC-L example of NPC Claire sending a tweet to the player, uses a new verb together with a new object. This example translates into XML as:

Listing 2: XML example of a NPC sending a tweet to the player.

```
<gamemove subject="0">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="1"/>
   <property key="content" value="Hi"/>
  </mediaObject>
 </verb>
</gamemove>
```

The XML structure is similar to the XML structure to create an NPC. However, the most outer tag now is the *gamemove* tag, indicating who is executing the game move. Under the *gamemove* tag, the *verb* tag can be found having the type "sendto" and a new tag within for the *mediaObject*. The *mediaObject* defines the type of object that will be sent. The *who* tag defines who the tweet is directed to and the *property* tag has an attribute *key* having the value *content* to define the message that will be tweeted.

Sequence

To translate sequential ATTAC-L game moves to XML, only two attributes need to be added to the XML structure:

Listing 3: Example of a sequence in XML.

```
<gamemove subject="0" id="tw1" next="tw2">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="1"/>
   <property key="content" value="Hi" />
  </mediaObject>
 </verb>
</gamemove>

<gamemove subject="1" id="tw2">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="0"/>
   <property key="content" value="Hello" />
  </mediaObject>
 </verb>
</gamemove>
```

The *gamemove* tags now have a *id* and *next* attribute instead of the *subject* attribute. The *id* attribute uniquely defines the game move; the *next* attribute defines which game move should be executed after the current game move. By adding these two attributes, no major changes are required to the XML structure. These two new attributes also support some control structures to happen, such as the *choice* control structure.

Choice

The *choice* structure in ATTAC-L uses the *id* and *next* attribute to define what should happen after the choice is made. An example can be seen here:

Listing 4: XML example of a choice.

```
<options id="choice1">
 <option next="opt1">Tweet "Hi"</option>
 <option next="opt2">Tweet "Hello"</option>
</options>

<gamemove subject="0" id="opt1">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="1"/>
   <property key="content" value="Hi" />
  </mediaObject>
 </verb>
</gamemove>

<gamemove subject="0" id="opt2">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="1"/>
   <property key="content" value="Hello" />
  </mediaObject>
 </verb>
</gamemove>
```

Much of the previously defined structures are re-used here. There is also the addition of the *options* and *option* tags that define the choice that can be made and specify which game move should happen when one is chosen.

Concurrency

Apart from a *choice*, *concurrency* is also possible to encapsulate two or more paths that can be followed concurrently. *Concurrency* requires one tag to be added that encapsulates multiple game moves:

Listing 5: Example of a concurrency in XML.

```
<concurrency>
 <gamemove subject="0">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="1"/>
    <property key="content" value="Hi" />
   </mediaObject>
  </verb>
 </gamemove>

 <gamemove subject="0">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="1"/>
    <property key="content" value="Hello" />
   </mediaObject>
  </verb>
 </gamemove>
</concurrency>
```

Use case

By combining the above XML listings, it is possible to translate the complete ATTAC-L use case to XML. The resulting XML to generate the NPCs can be found in Listing 1, while the rest of the scenario can be found in Listing 6.

Listing 6: Use case translated to XML.

```
<gamemove subject="1" next="con1">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="2"/>
   <property key="content" value="You better don't show your
   face tomorrow at school" />
  </mediaObject>
 </verb>
</gamemove>

<concurrency id="con1" next="con2">
 <gamemove subject="0">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="1"/><property key="content" value="Please
    shut up"/>
   </mediaObject>
  </verb>
 </gamemove>
 <gamemove subject="0">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="2"/><property key="content" value="Ignore her"/>
   </mediaObject>
  </verb>
 </gamemove>
</concurrency>

<concurrency id="con2" next="select1">
 <gamemove subject="1">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="0"/>
    <property key="content" value="You better watch out
    tomorrow, bad things will happen" />
   </mediaObject>
  </verb>
 </gamemove>
 <gamemove subject="1">
  <verb type="sendto">
   <mediaObject object-class="tweet">
    <who id="2"/>
    <property key="content" value="You better watch out
    tomorrow, bad things will happen" />
   </mediaObject>
  </verb>
 </gamemove>
</concurrency>

<options id="select1">
 <option next="opt1">Tweet "Bring it on"</option>
 <option next="opt2">Tweet"HAHAHA"</option>
</options>

<gamemove subject="0" id="opt1">
 <verb type="sendto">
  <mediaObject object-class="tweet"> <who id="1"/>
   <property key="content" value="Bring it on" />
  </mediaObject>
 </verb>
</gamemove>

<gamemove subject="0" id="opt2">
 <verb type="sendto">
  <mediaObject object-class="tweet">
   <who id="1"/> <property key="content" value="HAHAHA"/>
  </mediaObject>
 </verb>
</gamemove>
```

V. Scenario Code Generator

For the implementation of the ATTAC-L system in the game engine, a scenario code generator is created. This scenario code generator ensures that the correct scenarios and game moves are activated or executed based on the input and actions of the player. For example, in the use case when the player selects the option to send the tweet "Bring it on" to Kate, the scenario system generates a tweet back to the player.

Another goal of the scenario code generator is to hide the implementation details of the virtual world from the scenario writers. However, it is still necessary for the writer to provide the system with some basic information, for example, the positioning of game characters and the location of important objects.

A general overview of the scenario system is given in Figure 8.

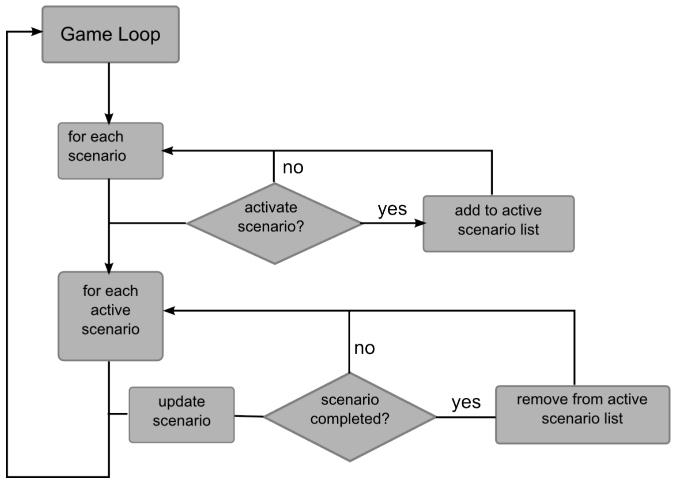


Fig. 8: Scenario system

The game loop is executed a number of times per second (typical games aim for sixty frames per second). At the start of each frame, the scenario system will run its update method which will update the active scenarios, check if a new scenario must be activated or remove a scenario from the active list. The update process of the active scenario executes game moves and in the case of sequential game moves also checks if a game move has been completed. In Figure 9, it is also shown that a game move can be executed on another thread, for example when an audio file is played. In that case the scenario update thread will check for the completion of the audio on regular time intervals. For short game moves (such as sending a tweet) the game move can be executed on the scenario update thread itself. A special case is the wait game move, which will disallow any other game move during its execution.

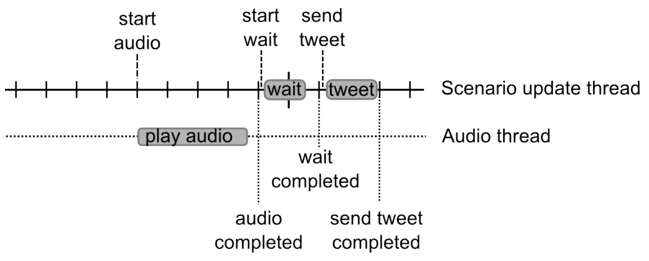


Fig. 9: Sequential game moves

For a concurrency block the situation is similar, but every game move will be executed on a separate thread. The update process of the concurrency block will then check for the completion of all the game moves that are children of this block. In the example in Figure 10 the concurrency block sets three characters on a path. The scenario will only continue when the three characters reach their distination. This allows a large amount of flexibility, for example the scenario will still work when one of the characters is placed in a different starting location.

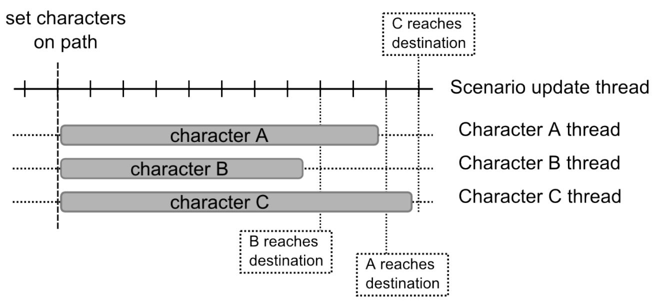


Fig. 10: Concurrent game moves

A scenario can be activated when any number of conditions is met. For example, it is possible to activate a scenario when a player enters a predefined area of the virtual world, such as a classroom. Another activation possibility is when a game character "sees" (defined by a field of view and range) another character or the player himself. Finally, it is possible to schedule a scenario for later activation, by defining either a relative or absolute time for the activation.

Once a scenario is activated, the game moves in the scenario are executed, and the scenario is updated each frame. During the update process, the state of the current game move is checked. If the game move has completed, the next game move will be started. For example, if audio is used in a game move, the next game move will only be executed once the audio clip has completed. The update process is somewhat different in a concurrency block. In such a block all game moves will be started simultaneously, and the system will check of all the game moves in the concurrency block have completed. If this is the case the next game move block will be executed.

VI. Resulting Game

The following screenshots clarify how the connection with the game world is made. The choice was made to create a 3D game, though the concepts explained here are independent of the type of game. As a result of the non-playable character creation process, shown in Figure 2, two characters are created in the virtual world, namely John and Kate, as shown in Figure 11. The female character in the center is the player avatar.

In the presented use case, the scenario starts when the player (the female character in front) is close to John. The non-playable character John receives a tweet from Kate and in this case the avatar responds without user input, see Figure 12.

After a couple of exchanges, the player is then presented with a choice, as defined in the options block in Listing 6. This is shown in Figure 13

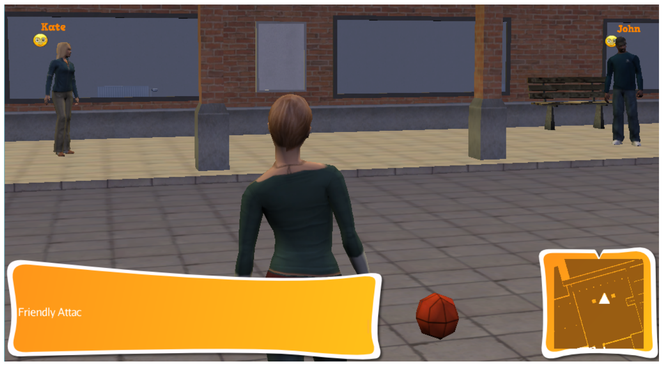
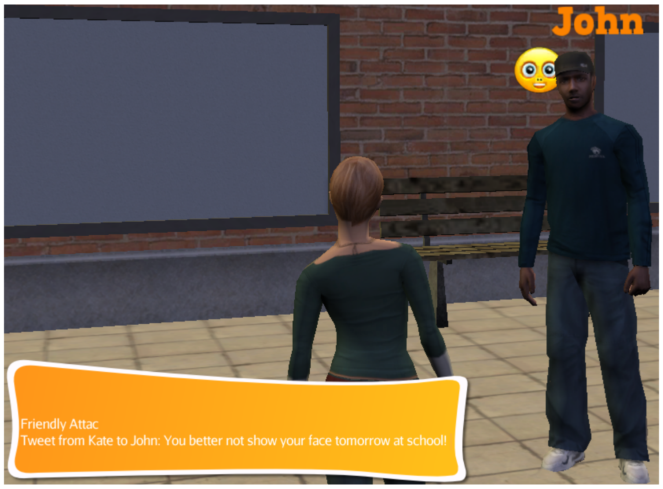
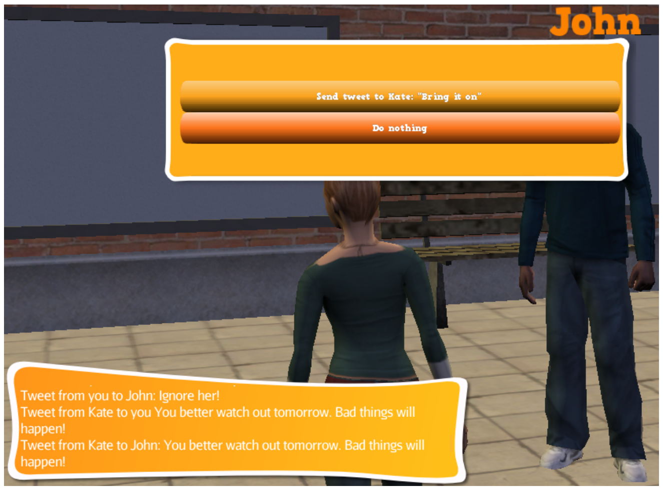


Fig. 11: In game - character creation



Friendly Attac
Tweet from Kate to John: You better not show your face tomorrow at school!

Fig. 12: In game - start of scenario

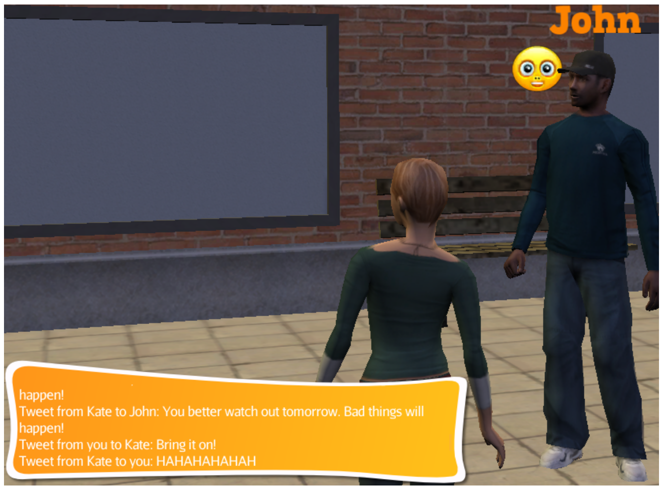


Tweet from you to John: Ignore her!
Tweet from Kate to you You better watch out tomorrow. Bad things will happen!
Tweet from Kate to John: You better watch out tomorrow. Bad things will happen!

Fig. 13: In game - choices

In the last screenshot (Figure 14) the last game move has finished, and the scenario is done.

In this scenario the choice was made to use the “seen by” condition, however this can be easily replaced by another condition.



happen!
Tweet from Kate to John: You better watch out tomorrow. Bad things will happen!
Tweet from you to Kate: Bring it on!
Tweet from Kate to you: HAHAHAHAHAH

Fig. 14: In game - end of scenario

VII. Testing Framework

With the aim of highly personalized virtual experience scenarios, it is critically important to create a suitable test framework for these scenarios. Throughout the game the player can select individual responses to situations within scenarios. These responses can change the outcome of the current scenario, and can also influence the unfolding of situations in later scenarios. However, during development it is also necessary to test a scenario in isolation. For example, if a scenario takes place after thirty minutes of gameplay, it is not feasible or practicle to play the game only to test one particular scenario. To tackle this problem, two testing solutions are provided: save games and rewind. Save games are important for the initial development of a scenario, while rewind is more important for the exhaustive testing of a given scenario.

The save game is a well understood mechanism to store the current state of the game. While developing and testing a particular scenario, the social scientist can simply create a save game that will store the internal states of the player and all the in-game characters. When a problem with a scenario is detected, the ATTAC-L scenario structure can be adapted, and the game can then simply be rerun from the saved game. The saved game is also defined in XML and will run silently before the 3D world is shown to the player. Today, this mechanism is allready fully implemented in the scenario code generator.

The choices of the player influence the logical flow of the scenarios. It is therefore necessary to test all the logical pathways (or as many as possible) before the game can be released. Restarting the game to follow another path in a given scenario is not efficient during this exhaustive test, so a mechanism must be provided to “rewind” the game. After the completion of a given scenario, the developer can rewind the game to the beginning of the scenario and follow other paths within the scenario. This rewind mechanism is ongoing work, but can be implemented by reusing the current save game mechanism. If rewinding is enabled during testing of a scenario, alle game state changes are saved in an XML definition file, and are linked to the game move element that caused the state change. When the developer rewinds the game, the game state changes are undone. The rewinding process includes resetting the next game move or concurrency block that will be executed.

VIII. Conclusion and Future Work

This paper presents a virtual game scenario generation process for serious games. By using this process it is possible for non-technical people to model virtual scenarios in the serious game. Technical details are provided on how the scenarios can be modelled in ATTAC-L and afterwards be translated in computer interpretable XML. This XML is then used to automatically build the scenario and game moves within the game engine so that it can be played. In order to test the scenario, a testing framework is also provided. Thanks to the presented tools and methods, it is possible to transform a written scenario into an in game virtual game scenario without having technical or game programming skills.

Future work consists of extending ATTAC-L. The current version of ATTAC-L has limited pre-defined verbs, resulting in a limited amount of pre-defined actions in the scenario code generator. In the future the vocabulary of verbs will be expanded, matching the story behind the serious game and allowing more actions to become possible in the game.

Acknowledgment

This work was funded by the IWT SBO Friendly ATTAC project (http://www.friendlyattac.be/), a project of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT).

References

[1] Friendly Attac. Available: http://www.friendlyattac.be/en/.

[2] L. K. Bartholomew, G. S. Parcel, en G. Kok, “Intervention mapping: a process for developing theory- and evidence-based health education programs”, Health Educ Behav, vol. 25, nr. 5, pp. 545-563, okt. 1998.

[3] Friendly Attac, “Computergestuurde interventies voor de bevordering van gezondheid bij jongeren: programmas op maat en het gebruik van videospelletjes”, http://www.friendlyattac.be/publicaties, 2013

[4] G. N. Yannakakis and J. Togelius, “Experience-driven Procedural Content Generation”, IEEE Transactions on Affective Computing, vol. 2, issue 3, pp. 147-161, 2011

[5] J. Togelius, G. N. Yannakakis, K. O. Stanley and C. Browne, “Search-based Procedural Content Generation: A Taxonomy and Survey, IEEE Transactions on Computational Intelligence and AI in Games, Special Issue on Procedural Content Generation vol. 3, issue 3, pp. 172-186, 2011.

[6] J. Luoma, S. Kelly, en J.-P. Tolvanen, “Defining Domain-Specific Modeling Languages: Collected Experiences”, Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling (DSM 04), 2004.

[7] F. Van Broeckhoven and O. De Troyer, “ATTAC-L: A Modeling Language for Educational Virtual Scenarios in the Context of Preventing Cyber Bullying”, IEEE 2nd International Conference on Serious Games and Applications for Heath, 2013.