serious game,
state machines, event system,
scripting language

Tomasz GRUDZIŃSKI[1], Robert MIKUSZEWSKI[1], Agnieszka SZCZĘSNA[1],
Mariusz SZYNALIK[1], Kamil URBANEK[1]

SCRIPTING LANGUAGE IN SERIOUS GAMES FOR PSYCHOLOGY

Computer and video games are more and more used for serous purposes not only for entertainment. In few last years we can observe very fast market development of serious games. They are used to education, simulation, research, health and therapy purposes.

This paper describes main features of serious games used for psychology. Specialists prepare scenarios of games with therapeutic points. To fast, easy and efficient implementing such scenario of game the scripting language can be used. In this paper the main features of the scripting language the Sciptarin to support implementing serious games for psychology were described.

1. INTRODUTION

Today many people with special focus on young population spend their time in front on computer playing games. Such games are often regarded as negative because of their violent content creating aggressive behavior, association with school failure and overweight. In opposite we can define other group of games which are serious games.

Serious games is a general term used for applications that are developed using computer game technology and game design principles but are used for non-entertainment purposes. We can say that they are *entertaining games with non-entertainment goals* [19]. The idea is that games could be used for more serious purposes such as education, simulation real world phenomenon and relations, increasing life quality through health, rehabilitation and therapy applications or raising interest to the problems in out global world ("Games for Change") [20]. An example can be "Darfur is Dying" an online game that is a simulation of life in a Darfur refugee camp or "Peacemaker" which targets the volatile Israel-Palestinian crisis to promote dialog and understanding. Another example is game "Travel in Europe". This is virtual world where people can enjoy challenging and engaging travels through European heritage [23].

This paper concerns game for health which are about 8% of all serious games [24]. In this group of serious games we can find following examples:

- health promotion games (for example "Re-Mission" game which help improving the lives of young persons living with cancer),
- games with information about diseases and treatment procedures,
- rehabilitation games (for example serious game for upper limb rehabilitation following stroke [22]),
- games with psychology content which are described in next section.

Full classification and catalog of serious games can be found on [21].

2. SERIOUS GAMES IN PSYCHOLOGY

Psychotherapy is an area in which innovative use of computer in the form of psychotherapeutic games may enhance patient compliance and offer new ways of treatment. This helps to attract and sustain the interest especially of children and adolescents which are groups that therapists often have difficulty engaging with. It is important to notice that the computer is a powerful medium because it is an accepted part of teenage culture. Such games could be attractive homework assignments between sessions, tools

[1] The Silesian University of Technology, Institute of Computer Science, ul. Akademicka 16, 44-100 Gliwice, Poland.

for structure therapy sessions, way for better communication and understanding parents with children. Results can be base of psychometric an personality tests.

Examples of games in psychology area are:

- "Treasure Hunt" for cognitive behavior therapy [16, 17].
- "Earthquake in Zipland" for family therapy to support children whose parents have divorced [18].
- "Personal Investigator" a game based on brief solution focused therapy [14, 15].

Other examples of serious games in this area are psychometric and personality tests [25] and games with guides to what act in difficult situations for example games for parents with information how to communicate with children.

Generally in psychology games users are represented by their avatar and can interact with the word and other game characters [19]. In game like in bibliotherapy [3] the story itself has a therapeutic component as much as it also evokes identification, empathy, resistance, opposition and disclosure of many confusing emotions. This goal oriented gaming can be used in goal oriented therapy methods (for example solution focused therapy). The game is play by scenario where user can observe and react for different situations. This act as powerful tools for helping users understand and reflect on their own behavior and give opportunity to learn from this virtual experience. Next thing is testing the user new abilities with new situations by different games goals. Games support self-reflection and behavior change through:

- providing detailed and objective feedback;
- presenting in virtual world (mostly 3D) the implication of choices;
- provoking emotional reactions;
- leading the user step-by-step through the decision making process.

Such scenario with therapeutic points should be prepared by psychotherapists. This is description of behaviors (artificial intelligence) of all characters (simulation actors) in game. To fast, easy and efficient implementing such scenario of game the scripting language can be used.

3. SCRIPTING IN GAMES

Scripting languages are widely used in gaming for many years [1, 11, 4]. The games' commercial success is often due to its ability to create alternative worlds and to entertain game players by allowing them to meaningfully interact with these artificial environments. As the hardware advances, it is becoming more and more often for players to concentrate mostly on the game content and depth of its story rather than its graphic performance. Therefore the scripting languages' lower efficiency – its main disadvantage – is steadily disappearing.

Among the most popular game titles there are virtually none that do not have any scripting language support. Depending on the game content and the designers' requirements the choice of the scripting language may be quite different. There are titles, like Eve-online [6], that use common-use scripting languages like Python (though Eve uses its special version, so-called *stackless python*). Some languages have been developed that are mainly used in games, though they are not limited to – among these the Lua language [9], used in World of Warcraft, Heroes of Might and Magic V or FarCry, should be mentioned. Then, there are games that have their own dedicated scripting languages like Unreal with its UnrealScript [13] or The Second Life with the Linden language [1].

A decision between implementing a new scripting language dedicated for the specific engine or game and using a proven and robust language that is already available is not always easy. For the idea behind this paper we present *The Scriptarin* language [8, 12] which is designed specifically to support developing serious games based on our own 3D Engine, Flexible Reality Simulation [7]. The decision we made was influenced mostly by the fact that we required complete control of how the language and its virtual machine works and what the features it offers.

4. THE SCRIPTARIN LANGUAGE

Scripting languages are extensively used to support AI in gaming [2, 5, 10]. *The Scriptarin* language was specifically designed to support writing programs operating in virtual reality simulations. Every program written in Scriptarin is a class of game objects often referred to as **entities**. It is said that every entity plays its part in the simulation, though not all of them must execute any code at all.

Scriptarin class is placed somewhere between a typical program and a class known from object-oriented languages such as C++ or Delphi. The Scriptarin makes it possible to inherit classes from other classes and so is quite similar to object oriented language, however, during the simulation each entity is a distinguishable, independent program and should be always treated as such. Scriptarin makes it possible to define a set of attributes defining class and its behavior – most of them (variables, constants, types, methods or operator overloading) widely known from common programming languages. Apart from these, the language offers some dedicated game supporting features like event handling, specific data types and - first and foremost - state machines.

Though game entities can be defined entirely in Scriptarin, it is obvious that scripting every basic function – like math routines for example – would significantly decrease game efficiency. Scriptarin takes advantage of routines written in the native code – every class or method can be declared as native and as such is mapped onto a class or method implemented in native code. Similar technique can be used when it comes to class fields (variables). Though native, these elements can be accessed and used in script code exactly in the same manner as script fields and methods making it easy to combine efficient, native code with high level, mainly AI-oriented, script routines.

5. USING SCRIPTARIN IN SERIOUS GAMES

Using the language to implement scenarios in games has been proven successful when the game DWARFS (http://dwarfs.artifexmundi.com/) was released. The game – although net being a Serious Game in fact – used Scriptarin quite intensively with all of its scenarios written purely in Scriptarin with only some base classes and low-level game engine written in native code.

The technical aspect of serious games is similar to typical games and in fact serious games are sometimes even simpler to code as they do not require such top-end graphics or depth. The Scriptarin language delivers two main features which make building scenarios very easy yet still maintaining complete control over the code – State Machines and Event System. These two features are described thoroughly in the following sections. Detailed information about all language elements can be found in Language Reference [8].

5.1. GAME ENTITY AS A STATE MACHINE

Every Scriptarin class is a finite state machine. The language offers structures to define a state along with some instructions to control the state code. A class does not have to define a state – some static entities do not require state code at all and can be declared without any. There is no limit in the number of states a class can declare and the language offers some straightforward techniques supporting state switching, entering and exiting. A simple Scriptarin state code is presented in example 1.

```
// logging loop iteration count in every iteration
state LogIt;
begin
  // entering state code
  init:
    var i: integer;
    i := 0;
    Log("Entering state");
  // main state code executed in every frame
  loop:
```

```
    i++;
    Log("Value of i: " + i);
  // exiting state code
  exit:
    Log("Exiting state");
end;
```

Example 1. An example of simple Scriptarin state code.

The Scriptarin language offers some specific techniques supporting state programming. Basically, code contained in the *loop* segment of the state is executed once per frame, while code within the *init* section is executed only once when entity enters the state and code within the exit section when entity exits the state. However, there are some special circumstances that can pause or even stop *loop* state code executing. These can be divided into three groups:

- **wait** commands – state code can wait for a specific time or wait until some specific condition is met (*boolean* expression is evaluated to *true*) before continuing. If the condition is specified, it is evaluated once per frame until it is true,
- **latent** calls - following the idea of UnrealScript [13] the Scriptarin language introduces the powerful mechanism of latent functions, routines that exit only after some specific condition has been met (animation has ended, a spell is cast, etc.). After calling latent function the state code pauses executing until the latent function exits. Latent functions must be declared as native and properly implemented in native code,
- **switch state** commands – an entity can go to another state or go to empty state (no state) by calling *goto statename* command, such command triggers the state switch independent of any other circumstances. The exit code of the current state is executed followed by the init code of the new state, if any applies. There is one more special state switching command, *goback*, which returns to the previously active state.

5.2. EVENT HANDLING

Another key feature of the language is event handling. An event occurs if an entity receives a message. A message is a collection of data with a given name that can be sent to a single entity or entity group. A message is always declared as the class attribute. Only the class which is aware of the message can respond to it. An entity class does not have to respond to a message, but if it wants to do so, a proper event handler must be declared.

The strong point of event handling is that a single class can declare multiple handlers for the same event. The only condition that must be met is that every handler must be declared within a different state. Additionally, a single event handler can be declared externally, beyond the state declarations – the one referred to as the global handler. When a message is received, the actual handler is determined depending on the current state of the entity. If available, a handler from the current state is called, if not – the global handler is. If none of these exists, message is ignored and nothing happens.

Moreover, the language introduces another useful mechanism – event gateways. A gateway is declared in the same manner as an event. It is called only when the specific message is received, just before the event call. Event gateway must have the same name as the event it refers to and of course must possess the same argument list. Gateway does not return a value and cannot be called directly. An example of the event and gateway declaration is given in example 2.

```
// accept the message only if the string "hello" is passed
gateway OnHello(AInvitationText: string);
begin
  if AInvitationText = "hello" then
    accept
  else
    reject;
end;
// respond to the event by posting "Hi!" to the sender
```

```
event OnHello(AInvitationText: string);
begin
  send sender.Hello("Hi!");
end;
```

Example 2. Event and gateway definition example.

Gateways are dedicated to support a good and clear programming style. When an entity receives a message, if a gateway is available, it must pass through the gateway before reaching event handler itself. The gateway can reject the message therefore not allowing it to reach event handler. The gateway should always exit with one of the two special commands – accept or reject. A class can define a single global gateway for many state-dependent event handlers or define many state-dependent gateways for the single, global event handler.

5.3. EXAMPLE

The serious game based on specialist scenario [26] founded on bibliotherapy with some therapeutic goals is planned to implement with use of the Scriptarin language. This game will have cognitive, behavior and developmental elements. It will concerns issues and problems of late childhood and adolescents. This will be some kind of guide with exercises where players can practice how to manage apathy, dejection, low selfesteem, anger, annoyance and stress.

To present the use of the Scriptarin language one scene where player learn how to manage anger and annoyance will be described. In this scene are two characters Player and Anger which are entities (objects) of Scriptarin class. Player has choice how to restrain Anger which demolish surrounding. Player and Anger have base class Character which consists of functions for defining main behaviors of characters. Example 3 presents how to define logic state transitions.

```
// const strings
const Q1_P = "Who are you? Why are you uprooting the trees?";
const Q2_P = "Why are you angry? What happened?";
const R1_P = "Don't worry, Anger, maybe they'll be found - and if not, you'll
collect new fruit. I can help you.";
const R2_P1 = "Can I make your hairdo? I can't see your eyes.";
const R2_P2 = "Now you look better.";
const R3_P = "Come on, Anger, let's play ball.";
const R4_P = "What are you doing?! You're going mad!!!";
const R1_A = "I'm Anger, I'm Anger! I can do whatever I want! I can demolish
what I want, I can scream as loud as I want to!";
const R2_A = "Someone has eaten up my fruit, which I'd been collecting the whole
day yesterday! I'm terribly angry!";
const R3_A = "Now I'm better.";

class Anger expands Character;
...
// Global gateway; visible in all states.
gateway OnHearPlayer(AMsg: string);
begin
  if AMsg in [Q1_P, Q2_P, R1_P, R2_P1, R2_P2, R3_P, R4_P] then
    accept
  else
    reject;
end;
// Initial state: Anger keeps shouting and breaking trees
state S_Angry; initial;
begin
  // Event handled only in the A_Angry state
  event OnHearPlayer(AMsg: string);
  begin
    // Reply to the Player with respect to what he/she said
    case AMsg of
```

```
      // Continue conversation in S_Angry state
      Q1_P:
        begin
          send Player.HearAnger(R1_A);
          Shout();
         end;
      Q2_P: send Player.HearAnger(R2_A);
      // Get better if the Player wants to help
      R1_P: goto S_Agreement;
     // Nod as a sign of agreement
     R2_P1: Nod();
     // Get better if the Player wants to help
     R2_P2: goto S_Agreement;
     R3_P: goto S_Agreement;
     // Attack if the Player calls you mad
     R4_P: goto S_Fight;
  end;
  init:
  loop:
    // Keep demolishing the forest
    BreakATree();
  exit:
end;

// Anger calms down
state S_Agreement;
begin
  init:
    // Say it's fine
    send Player.HearAnger(R3_A);
  loop:
    // Keep smiling
    Smile();
  exit:
end;
// Scuffle
state S_Fight;
begin
  init:
    // Attack
    send Player.GetAttacked();
  loop:
    // Keep attacking
    Fight();
  exit:
end;
class Player expands Character;
...
// Global gateway; visible in all states.
gateway OnHearAnger(AMsg: string);
begin
  if AMsg in [R1_A, R2_A, R3_A] then
    accept
  else
    reject;
end;
// Initial state: the Player keeps asking questions
state S_Curious; initial;
begin
  // Event declared only for the S_Curious state
  event OnHearAnger(AMsg: string);
  begin
    // Continue conversation with Anger with respect to his replies
    case AMsg of
      R1_A: send Anger.HearPlayer(Q2_P);
```

```
      // Give the Player a choice
      R2_A: goto S_PlayerInput;
  end;
  init:
    // Ask initial question
    send Anger.HearPlayer(Q1_P);
  loop:
  exit:
end;
// The Player's choice: making proposals to Anger
state S_PlayerInput
begin
  // React to Anger nodding as a sign of agreement
  event OnAngerNods();
  begin
    // Continue helping Anger
    MakeHairdo(Anger);
    send Anger.HearPlayer(R2_P2);
  end;
  // React to Anger attacking
  event OnGetAttacked();
  begin
    // No more talking, let's fight...
    goto S_Fight;
  end;
  init:
    // Id of the Player's choice
    var P_Choice: integer;
    // Get the Player's choice
    P_Choice := GetPlayerInput();
    // Continue conversation with respect to the Player's choice
    case P_Choice of
      1: send Anger.HearPlayer(R1_P);
      2: begin
           // Approach Anger
           WalkTo(Anger.Position);
           send Anger.HearPlayer(R2_P1);
         end;
      3: send Anger.HearPlayer(R3_P);
      4: send Anger.HearPlayer(R4_P);
   loop:
   exit:
  // Scuffle
state S_Fight;
begin
  init:
  loop:
    // Keep fighting
    Fight();
  exit:
end;
end;
```

Example 3. Event and gateway definition example.

6. SUMMARY

This paper describes main features of serious games used for psychology. Specialists prepare scenarios of games with therapeutic points. To fast, easy and efficient implementing such scenario of game the scripting language can be used. The main features of the scripting language the Scriptarin like State Machines and Event System can support implementing serious games for psychology. The serious

game based on specialist scenario founded on bibliotherapy with some therapeutic goals is planned to implement with use of the Scriptarin language.

BIBLIOGRAPHY

[1] BRASHEARS A., MEADOWS A., ONDREJKA C., SOO D., Linden Scripting Language Guide, Linden Lab, 2003.
[2] COMBS N., ARDOINT J.L., Declarative Versus Imperative Paradigms in Games AI, Independent, ILOG S.A., 2004.
[3] MOLICKA M., Bajki terapeutyczne, Media Rodzina, 1999.
[4] DELOURA M., Game Programming Gems Vol. 1, 2, Charles River Media, 2000–2001.
[5] ELDAWY M., Interfacing Between Game Engine and Scripting Languages, Report for Academic ADL Co–Lab, 2006.
[6] EVE–ONLINE official web page, http://www.eve–online.com , 2003–2007.
[7] GRUDZINSKI T., MIKUSZEWSKI R., Splitting it into Components: Designing an Interface–Oriented 3D Game Engine, Learning with Games 2007 Conference Proceedings, 2007.
[8] GRUDZINSKI T., The Scriptarin – Language Reference, www.artifexmundi.com/files/script.pdf, 2006.
[9] LUA official web page, http://www.lua.org/ , 1993–2007.
[10] MCNAUGHTON M., CUTUMISU M., SZAFRON D., SCHAEFFER J., REDFORD J., PARKER D., ScriptEase: Generating Scripting Code for Computer Role–Playing Games, www.cs.ualberta.ca/~script/docs/ASEdemo2004.pdf , 2004.
[11] ONUCZKO C., CUTUMISU M., SZAFRON D., SCHAEFFER J., MCNAUGHTON M., ROY T., WAUGH K., CARBONARO M., SIEGEL J., A Pattern Catalog for Computer Role Playing Games, EUROSIS–ETI, 2005.
[12] THE SCRIPTARIANS official web page, http://www.artifexmundi.com/scriptarians , 2007.
[13] SWEENEY T., UnrealScript Language Reference. http://unreal.epicgames.com/UnrealScript.htm, 1998.
[14] COYLE D., MATTHEWS M., SHARRY, J., NISBET, A., DOHERTY G., Personal Investigator: A Therapeutic 3D Game for Adolescent Psychotherapy, International Journal of Interactive Technology and Smart Education, 2, 2005, pp. 73–88.
[15] COYLE D., MATTHEWS M., Personal Investigator: a Therapeutic 3D Game for Teenagers. Presented at the 'Social Learning Through Gaming' Workshop ACM CHI'04 Conference on Human Factors in Computing Systems, 26 April 2004, Vienna, Austria.
[16] BREZINKA, V., Treasure Hunt – A Serious Game To Support Psychotherapeutic Treatment Of Children. In: Andersen, S.K. et al. (Eds.), eHealth Beyond the Horizon – Get IT There, Studies in Health Technology and Informatics, Vol. 136, IOS Press 2008, pp. 71–76.
[17] BREZINKA, V., HOVESTADT, L., Serious Games can Support Psychotherapy in Children and Adolescents. In: Holzinger, A. (Ed.), HCI and Usability for Medicine and Health Care, Lecture Notes in Computer Science 4799, Springer, Berlin 2007, pp. 357–365.
[18] Therapy Computer Game for Children of Divorce, http://www.ziplandinteractive.com/.
[19] MAGNENAT–THALMANN N., KASAP Z., Virtual Humans in Serious Games, CW '09: Proceedings of the 2009 International Conference on CyberWorlds, IEEE Computer Society, Washington, 2009, pp. 71–79.
[20] SCHREINER K., Digital Games Target Social Change, IEEE Computer Graphics and Applications, Vol. 28, No. 1, 2008, pp. 12–17.
[21] A Collaborative Classification of Serious Games, http://serious.gameclassification.com.
[22] BURKE J. W., MCNEILl M. D. J., CHARLES D. K., MORROW P. J., Serious Games For Upper Limb Rehabilitation Following Stroke. VS–GAMES '09: Proceedings of the 2009 Conference in Games and Virtual Worlds for Serious Applications, Washington, 2009.
[23] Travel in Europe project, http://www.tieproject.eu/.
[24] RITTERFELD U., CODY M., VORDERER P., Serious Games Mechanisms and Effects, Routledge, 2009.
[25] MEMARZIA K., Measuring Behavioural Traits and Psychometric Testing Using Gaming Technology, ITEC 2008 Conference.
[26] WIETESKA A., Psychology Serious Game Scenario, Report, Silesian University, Katowice, 2010.