A 2D iPhone Game Framework

Lenka Pitonakova
contact@lenkaspace.net
University of Sissex 2009

Abstract

The project reviews some of the existing design patterns and available algorithms used in iPhone games development. The application created is a template project for a 2D iPhone game. Issues including efficient rendering, extendibility and objects persistence are discussed and algorithms for dealing with these issues are devised.
The core ideas are based on the Model-View-Controller design pattern and the subsumption architecture which are combined together to provide behaviour-based modules that form game world objects.

Keywords

iPhone development, 2D games, Model-View-Controller, factory pattern, decorator pattern, subsumption architecture

Contents

1. Introduction

1.1. Games Architecture Background

There are the following components usually present in games: game application delegate, game world (a wrapper object for all other objects), game environment (background, terrain, etc.) and game objects (also often referred to as 'sprites'). Sprites can be static (e.g. rocks) and dynamic (e.g. player's units, NPCs, etc.). In an object-oriented architecture, one or more classes can represent every component. Objects like Game World usually own instances of other objects like Sprites or Terrain. Lower-level objects use pointers to communicate with their wrapper object and with each other.

A game runs at a certain frame rate. On the iPhone, animation with OpenGL ES usually requires 60 frames per second (Mark D. and LaMarche J., 2009, p 454). Each class needs to update its state and render itself every frame. There are usually Update and Render methods available in each class, although in some cases the Update method can be empty. Higher-level classes first run their own routines and then call Update and Render methods of lower-level classes. The game application delegate uses a timer to determine when to start the next frame and starts the Update-Render chain each time.

As far as rendering goes, any rendered object usually needs to have an instance variable that refers to a texture used for that particular object. Since it is convenient to store textures in an array (Section 2.3), the instance variable corresponds to an index of such an array. Figure 1 shows a UML diagram of a general game architecture.

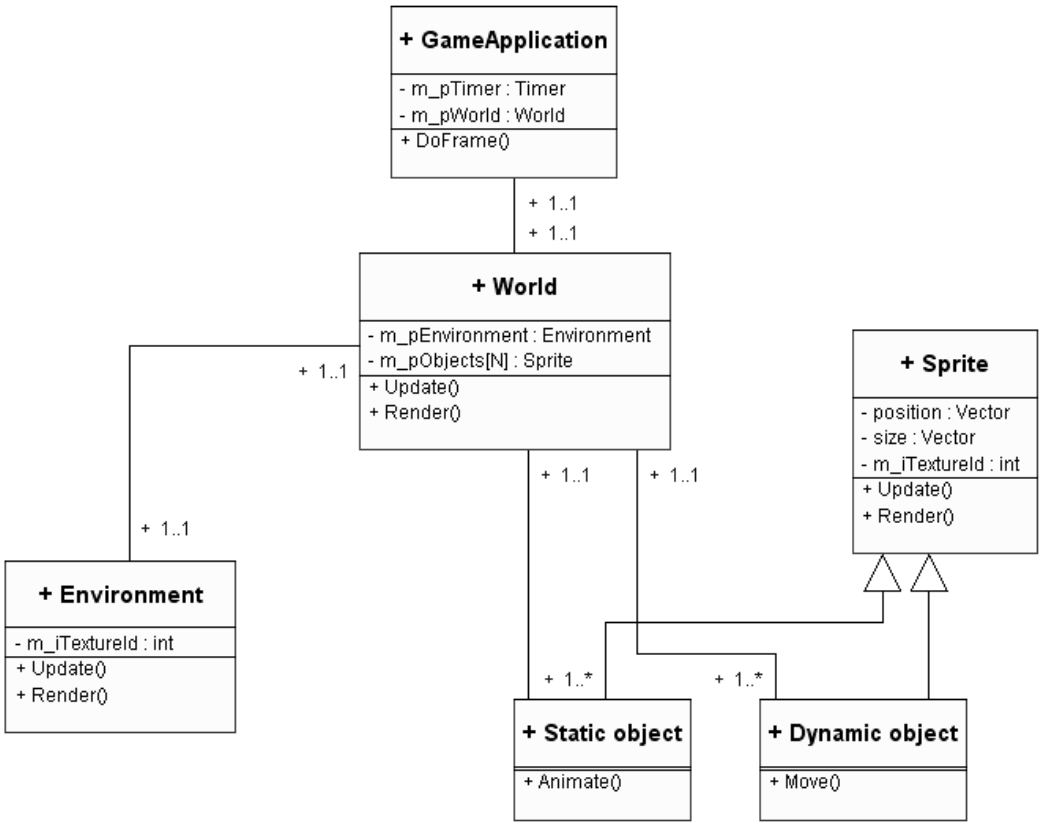

Figure 1: UML of a general game architecture

1.2. Aims and Objectives

The aim of this project is to create a template project suitable for development of iPhone 2D games that have a world and some static and dynamic objects and controllable units. The world should be viewed from the top. Examples of a static and a dynamic object are a rock and a selectable object.
The following objectives have been set for the template project:

1. Efficient rendering

The device's memory and computation time need to be saved as much as possible due to general iPhone memory and processing constraints (Mark D. and LaMarche J., 2009, p 7).

2. Modularity and extendibility

A developer should be able to add behavioural modules to dynamic objects without affecting other modules. By 'modules' we understand e.g. movement, rendering, obstacle avoidance, etc.

3. Application persistence.

The application needs to remember its state when it is closed and resume it when it is opened again. The template project has to provide a clear idea to a developer on how to achieve this. Archiving of template objects needs to be implemented.

4. Basic functionality

The dynamic objects can be selected and controlled by the user. They should be able to accept orders and execute tasks they recognize.
Furthermore, there should be units that can move when ordered so by a user and interact with each other by creating a new unit when they meet. The movement function was chosen because it is a basic property of any game units and it usually involves a lot of testing and adjustments as well as vector computations. The 'breeding' function was chosen to show how new modules with their own instance variables can be added into the project.

2. Rendering

2.1. The Starting Point

The starting point for this application is a model program provided by Fothergill A. (2009). He extended Apple's standard OpenGL ES rendering template and added methods for loading images into textures, processing them and rendering a texture of a certain size and on a given position.
All functions are included in the EAGLView and Texture2D classes originally provided by Apple and extended by Fothergill. In the application created for this coursework the EAGLView was used for notifying classes about user interaction and the Texture2D class was extended in order to suit the requirements. The ApplicationDelegate and EAGLView classes are not listed in this paper since they do not have any original author's code. The Texture2D class is listed in Appendix A.

2.2. Saving of the Computation Time

Today's graphics are on a considerably advanced level. Thanks to OpenGL and its Cocoa counterpart OpenGL ES, games can use complicated geometry and even more complicated textures. This however comes at the cost of the computation time. In OpenGL the whole screen gets erased and all objects and textures are rendered again each frame. A game with animation needs about 60 frames per second so that animations appear smooth (Mark D. and LaMarche J., 2009, p 454). This means that rendering of the whole scene needs to happen 60 times per second. Therefore, it is necessary to render only what is currently visible to the player. This can be achieved by putting a test into the Sprite's Update function:

```
if (m_ptPos.y < -SCR_H - m_ptSize.y/2 || m_ptPos.y > SCR_H + m_ptSize.y/2) {
        m_bShouldRender = false;
    } else if (m_ptPos.x < -SCR_W - m_ptSize.x/2 || m_ptPos.x > SCR_W + m_ptSize.x/2) {
        m_bShouldRender = false;
    } else {
        m_bShouldRender = true;
    }
```

Sprite's position is checked against the screen dimensions. There is a Boolean class variable that is used in the rendering method to test whether the Sprite should render or not. Each Sprite has a Render and DoRendering method. The first one is called by a wrapper class in the game loop, checks the Boolean variable and calls the DoRendering method. Any subclass only needs to overwrite the DoRendering method where the real rendering happens and can keep the test from its super class intact (OODesign: Decorator).

2.3. Saving of the Memory

The application designed by Fothergill A. (2009, p 72) uses the LoadTextureImage function in the application's delegate to load images into an array. The loaded images are used by rendering functions. This means that every image, even a small icon, needs to have a separate texture and a position in the array. An application with many small icons would have to have a big array that would consume memory, since one element of the array makes space for a texture of up to 1024x1024 pixels.

Therefore, Fothergill provides a function GrabSpriteSet in Texture2D that is able to take a single image and slice it into a number of rectangular textures, which are separately loaded into the array. However, the shapes of the slices have to be the same and individual array slots for each slice still need to be created. Each slice's size has to be adjusted according to the largest slice that is cut out form the texture and blank spaces are necessarily created.

The function added to Texture2D for this project is DrawSpriteAtWithUV(int a,CGPoint pos, CGFloat rot, CGPoint uCoordins, CGPoint vCoordins). It is able to render only a part of a texture based on the given U and V texture coordinates with values between 0 and 1. Both uCoordins and vCoordins are 2D vector parameters where the x (first) component specifies the beginning and the y (second) component the end of a slice in a given dimension. Figure 2 shows a texture that can be sliced into smaller images. To specify e.g. image of red unit on the right, U=[0.5,1] V=[0.25,0.5].

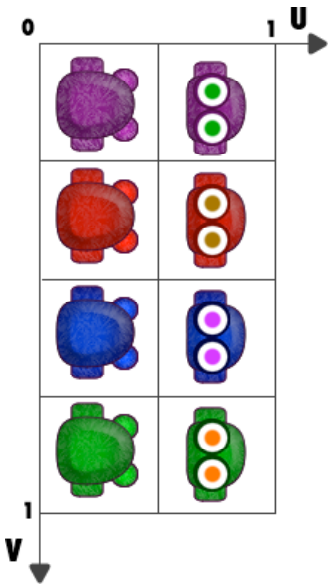

Figure 2

The function first calculates how big slices of UV mesh (uStep and vStep) are given the UV coordinates and which slice the given coordinates correspond to (uPos and vPos). Texture UV coordinates and dimensions are normally stored in an array. A specified texture from this array is copied to another temporary array and its UV coordinates and width/ height dimensions are adjusted and later applied in rendering (for full details please refer to Appendix A). This leaves the original texture intact and usable again with different UV coordinates.

The new UV coordinates are calculated as:

```
for (int i=0;i<8;i++) { // the array of UV coordinates has 8 elements which represent 4 corners of the texture
    //------------ copy UV coordinates from the original texture:
    textCoords[i] = gSprite[spr[a]].tverts[i];
    if ( (i)%2 == 0) {
        textCoords[i] = textCoords[i]*uStep + uPos*uStep*gSprite[spr[a]].tverts[2]; // U coordinates
    } else {
        textCoords[i] = (vStep*textCoords[i]) + vPos*(vStep*gSprite[spr[a]].tverts[1]); // V coordinates
    }
}
```

The size of the texture also needs to be scaled down, by the same amount as the UV coordinates. The uStep and vStep variables are reused:

```
for (int i=0;i<12;i++) { // the array of XYZ texture coordinates has 12 elements which represent 4 corners in 3D.
    //---------------- copy dimensions of the original texture
    textVerts[i] = gSprite[spr[a]].verts[i];
    if ( (i)%3 == 0) {
        textVerts[i] *= uStep; // width (X) coordinates
    } else if ((i-1)%3 == 0) {
        textVerts[i] *= vStep; // height (Y) coordinates
    }
}
```

The array indexes of U, V, X and Y elements were found experimentally.

This approach has the following advantages:

1. Shape of a slice does not have to be the same as of other slices, because the UV texture coordinates can be set for each slice individually. This means that maximum texture space is utilized.
2. The array of textures is significantly smaller than in the case of the GrabSpriteSet method. Only one texture is loaded into the array and is used for all slices with different UV coordinates. The approach provides a faster solution for applications with rich graphics. Even though the computation needs to happen each frame, accessing a busy memory is usually slower than running two for loops.

2.4. Rendering and Selection of Dynamic Objects

As it was mentioned above, there are two types of objects - static and dynamic - and both inherit from Sprite. Static objects simply render themselves using Sprite's DoRendering method. The class responsible for rendering of dynamic objects is DynObjView which inherits from NSObject. It does not store any rendering-related class variables and all parameters like texture ID, object's position and the UV coordinates are passed to its RenderAt method from UnitController, a subclass of Sprite (more details in Section 3.2.)

The RenderAt method checks whether a unit has been selected (the selection is handled by DynObjView itself) and adjusts its U coordinates accordingly (default U coordinates of each unit correspond to its unselected state). Figure 2 shows that the texture for units has been designed so that individual unit colors are under each other and a texture slice on the right is used when a unit is selected.
The RenderAt method then calls the DrawSpriteAtWithUV with appropriate UV coordinates. The decision about the V (horizontal) coordinate is made in UnitController 's init method since color corresponds to unit's type managed by the Controller. On the other hand, the U coordinate is determined in the RenderAt method since DynObjView handles selection and deselection. Please refer to Appendices C and F where both methods are listed. The selection process shown in Figure 5, Section 3.3.

There can only be one unit or dynamic object selected at a time in the current implementation. When user touches the screen all dynamic objects check whether they have been clicked on. If touch's position corresponds to a position of one of the objects, it deselects a selected one and selects itself. The implementation of selection is listed in

Appendix C, HandleTouchesBeginAt: method. According to principles of the decorator pattern (OODesign: Decorator), the method calls either UserStartTouchInside: or UserStartTouchOutside: method depending on whether a unit has been selected or not. Subclasses can overwrite the two methods individually, leaving the basic mechanism of the HandleTouchesBeginAt: intact.

2.5. Testing: Rendering of Units

In the following test the texture from Figure 2 was used to render all units. Textures for selected and unselected units were tested by plotting four different units into the world and selecting them individually. The units have the U coordinates set to [0,0.5] by default (i.e. when they are not selected). The following V coordinates are set in UnitController's SetupSelf: method (Appendix F): purple: V=[0,0.25], red: V=[0.25,0.5], blue: V=[0.5, 0.75], green: V=[0.75, 1]

The testing confirmed correct implementation of the DrawSpriteAtWithUV method. Further tests were made with different slice dimensions including U=[0,0.5] V=[0,0.5], U=[0.5,0.6] V=[0,0.5], U=[0.8,1] V=[0.5,0.6] etc.

Figure 3 shows the individually selected units under different rotations (green: 0°, purple: 90°, red: 180°, blue: 270°)

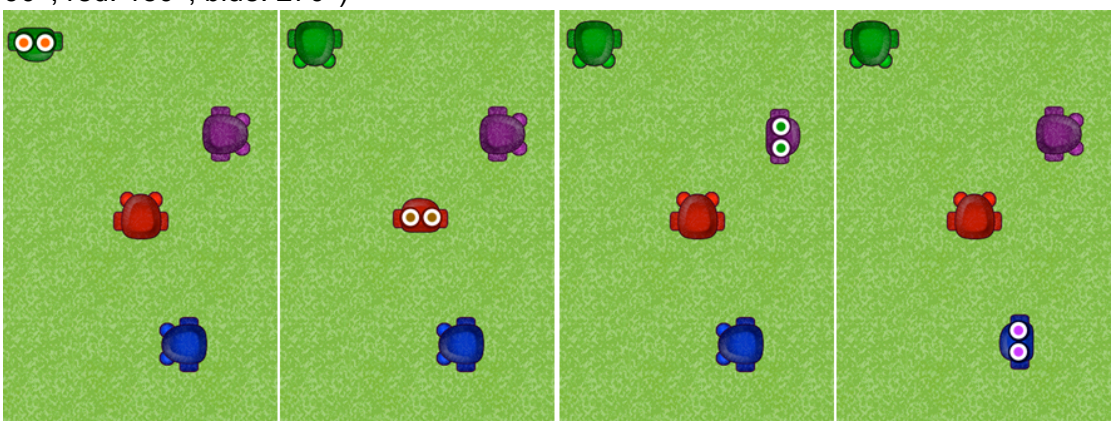

Figure 3: rendered units

3. Application of the Subsumption Architecture and the MVC Design Pattern

3.1. Background

The subsumption architecture was invented by R.A. Brooks (1991) and is inspired by biological systems. Brooks proposed that architecture for building robots should be behaviour-based to produce bottom-up logic (in comparison with classical AI's top-down approach). Such a robot would consist of independent modules, each responsible for only a part of its behaviour, e.g. creating snapshots from a camera, image recognition, steering adjustment, etc. As it was shown in other work of Brooks and others (Horswill I. D. and Brooks R. A., 1988; Franceschini N. et. al., 1992; Mackworth A.K., 1992; Webb B., 1996), the

subsumption architecture provides reliable and to some extent emergent behaviour. Also, adding further behaviours into a robot is easier than in classical AI.

A similarly modular approach is taken in the MVC (Model-View-Controller) design pattern recommended by Apple's developer guidelines (Apple: MVC). A View is responsible for rendering and interaction with the user, A Model stores data and a Controller provides an interface between the two and controls flow of an application. In other words, the Controller decides what to do and the Model and the View do low-level actions and calculations. The main advantage of the MVC pattern is programme extensibility. Methods added into Model or View are independent of each other and the changes are therefore manageable. Furthermore, additional logic and more complex behaviours can be added into the Controller, which does not affect the functionality of the Model or the View. Each of the three components (especially the Controller) can even be substituted for another class (e.g. predator and prey would only differ in implementation of the Controller).

This project implements the MVC design pattern that borrows behaviour-based approach from the subsumption architecture.

3.2. Implemented Design

In the basic implementation, a world had a static object, a dynamic object and units that could move. Based on differences in the functionality, individual behavioural modules and actions were identified according to the subsumption architecture principles:

- **Interaction with the user:** display self, select self, accept user orders
- **Memory:** store and retrieve basic characteristic, store and retrieve a given task (commonly referred to as 'store details')
- **Movement:** move to a location
- **Provide interface with the application:** archive models, handle current frame

The different behavioural modules were then divided into Models, View and Controller (Figure 4).

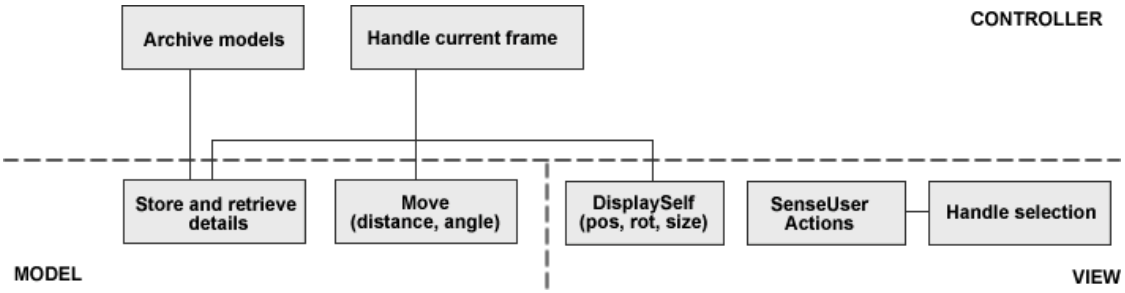

Figure 4: MVC behavioural modules

The first two categories are common for all dynamic objects since they can be selected. A basic Memory-View-Controller structure is therefore required.
Since there should be only one View (Apple: MVC), all user interaction has been made a part of the DynObjView class. Similarly, there is only one Controller and all higher-level behaviours are represented by methods in the DynObjController class. DynObjMemory stores and retrieves dynamic data (position, rotation, etc) and a current task (in the current

implementation there are no tasks available to all dynamic objects though). All Models and the View are given tasks from the Controller and have a pointer to the Controller to be able to call its appropriate notification functions.

Additional behaviour of units is movement. Subclasses UnitView and UnitMotor were created to extend Dynamic Object. UnitView overwrites some of the DynObjView's methods and UnitMotor inherits from DynObjModel to provide additional functionality. Also, units have their own controller which inherits from DynObjController. A UML diagram of the application is shown in Appendix M.

3.3. The Movement Algorithm

Units use their own UnitView class that inherits from DynObjView and overwrites the HandleTouchesEnded method. When user touches the screen, the overwritten method checks whether a unit has been selected and whether the latest touch did not selected another dynamic object. If both are true, it is understood as the MOVE order:

```
- (void) HandleTouchesEndedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_{
    if (m_bSelected && !GetIsAnotherObjectSelected(m_pController.m_iId)) {
        [m_pController UserOrder:UORDER_MOVE atLocation:pos_ withObject:NULL];
    }
}
```

The View calls DynObjController's UserOrder: atLocation: withObject: function with order identifier UORDER_MOVE. The order identifiers are stored as enumeration data type for better understanding of the code (Appendix J). The Controller uses the Memory module (DynObjMemory class) to store the current task and checks for an existing task in its Update function. In the implementation without any obstacle avoidance, the Controller simply passes the location of the remembered task to the Motor (UnitMotor) that handles the movement itself. Moreover, DynObjMemory checks whether a task has been completed each frame in its own Update method. If a task is completed (in the case of movement a unit reaches a target location), the memory notifies the controller by calling its FinishedTask: method (a general implementation in DynObjController sets current task to none and the unit to an idle state) The whole process is shown in Figures 5 and 6 and the code is listed throughout Appendices B-H.

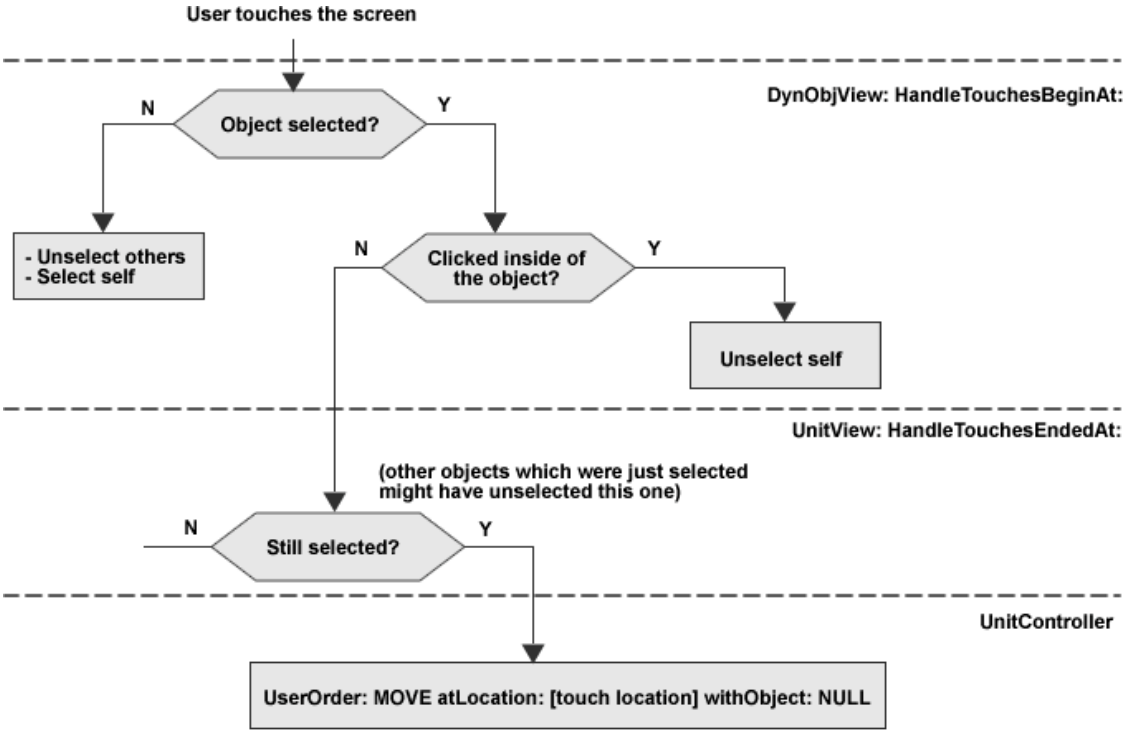

Figure 5: Handling of user's touch

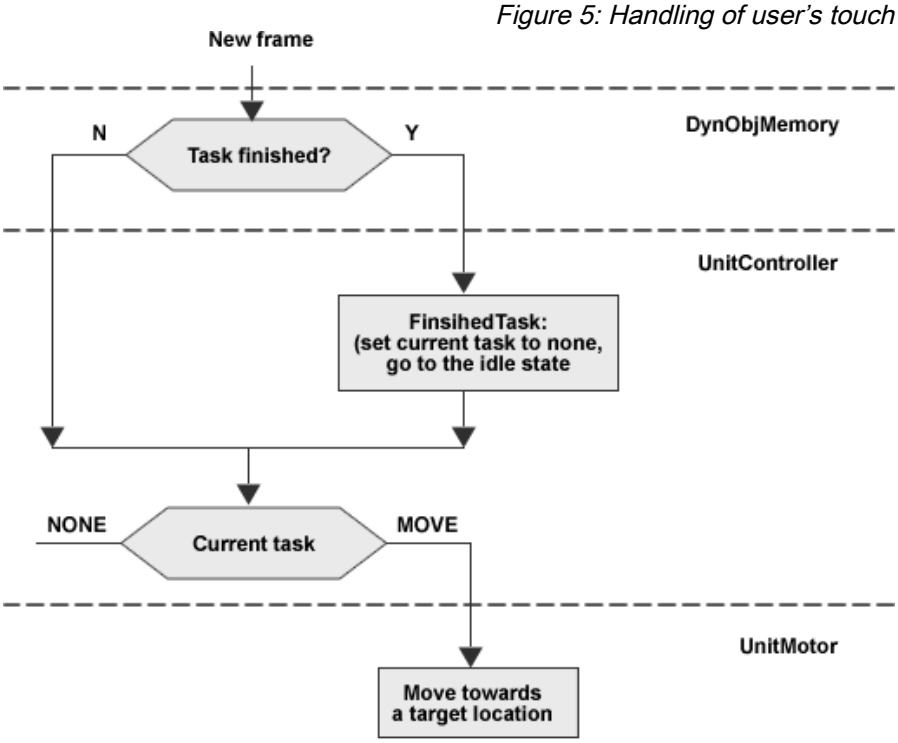

Figure 6: One frame of the Update loop

The movement algorithm implemented in UnitMotor's MoveTowards: method is based on the C++ algorithm by (D.M. Bourg and G. Seeman, 2004, p 16-19) and was rewritten into Objective-C along with necessary implementation of the Vector class (Appendix K) provided by the authors in C++ (2009, p 349 - 358).

A vector localVec is created from the unit to the target location and it is converted to the unit's local coordinate system. Unit's speed is adjusted according to the distance of the target, which is represented by localVec's y coordinate, as:

```
speed = 3 + fabs (localVec.y / 50)
```

This keeps a minimum speed of 3 and makes the unit move faster when the target is further away. LocalVec is then normalized so that only its direction is kept. Steering of the Motor is adjusted according to localVec's x coordinate which represents left and right side of the unit (-x for left, x for right, the unit's origin is at localVec.x=0):

```
turnByAngle = localVec.x*5
```

Complete Objective-C implementation of the movement algorithm can be found in Appendix H, method MoveTowards: (CGPoint)loc_.
A problem can occur when user clicks exactly behind a unit, i.e. localVec.x = 0. In this case the algorithm needs to distinguish between clicking in front and behind the unit. LocalVec's y coordinate is used as follows:

```
//------------------ test if user clicked along the local horizontal axis:
    if (localVec.x < 0.1 && localVec.x > -0.1) {
        //----------- test if click was behind:
        if (localVec.y > 0) {
            //------- make a big turn:
            [self TurnBy:45];
        }
    }
```

3.4. Testing: Movement

The movement was tested by selecting individual units and clicking on an empty terrain, units themselves and other units. Acceptance radius of 25 pixels around a target location needed to be set to prevent units circle around a point when their target location was very close to their original location. Testing was done multiple times with all four units.

The units always reached a target location with the set acceptance radius and changed their turning angle and speed according to the relative target's position. When user clicked on a selected unit, it got unselected and further touches outside of its bounds did not affect it. When a unit was selected and another unit was clicked on, the second one got selected and the first one stayed on its place.
The testing proved that the movement algorithm implementation was sufficient and correct.

4. Application persistence

One of the objectives set in Section 1.2. was the ability of the application to remember its own state. In Objective-C this can be achieved using property lists (arrays which are written into a file when application closes) or object archiving (ability of a class to encode and decode its objects into serial data and store the data) (D. Mark and J. LaMarche, 2009, p. 333-350).
Archiving is a recommended technique for persisting model objects (Apple: Models). It allows objects not only to be saved but also copied and transferred to another application (Apple: Archiving).

4.1. Models and Archiving

Static objects do not need to be archived since they do not change and can simply be instantiated each time an application opens. On the other hand, dynamic objects change their state and therefore need to be able to store and retrieve it.
Since all data about dynamic objects is stored in their Models, only the DynObjModel class needs to implement archiving. When an object it archived its wrapper needs to create an NSKeyedArchiver/ NSKeyedUnarchiver instance which encodes (saves)/ decodes (retrieves) an object. When encoding, an object's encodeWithCoder: method that handles archiving of instance variables is called. When decoding, a wrapper object needs to check whether a particular file exists and if so a dummy object of a decoded class is created using the initWIthCoder: method that handles decoding of instance variables. The wrapper then moves individual variables from the dummy object into an existing one (D. Mark and J. LaMarche, 2009, p. 343-344). Thus the wrapper class needs to know class of an archived object and its instance variables.

However, if developers are allowed to add additional Models with their own instance variables and classes, a factory pattern which lets subclasses handle their initialization needs to be used (OODesign: Factory) and general methods for encoding and decoding are needed in the wrapper object's class (DynObjController). The general methods assume that any encoded object is of DynObjModel class which implements the encodeWithCoder:, initWithCoder: and SetSelfWith: methods (Appendix D). Individual subclasses of DynObjModel overwrite these methods with their own instance variables.

Each Model subscribes to be notified when the application is closing in its initialisation method. Upon closing, a Model calls DynObjController's EncodeModelObject: usingKey: method that creates an NSKeyedArchiver and calls appropriate encoding method of a Model. Similarly, decoding of an object needs to be called from the wrapper when an object is instantiated. To avoid adding more lines of code to the Controller for each new class, each Model calls its [super initModelWithController: encodingKey:] method. The method checks whether a file exists (using the encodingKey parameter) and calls the general DecodeModelObject: usingKey: method of DynObjController, passing itself as a parameter. The Controller creates a dummy object of the given subclass (subclass returns a instance of itself its own implementation of the intiWithCoder: method) and calls its SetSelfWith: method that passes a decoded dummy object into a Model instance and lets the Model handle copying of instance variables (Appendix D). The Controller no longer needs to know about the subclass and variables of the encoded object.
The archiving keys need to be unique for each Model of each object. Thus an object's ID (index of dynamic objects array stored by the controller) is added to each Model's key.

Since dynamic objects can be added to the world when the application is running, the World stores an array of object identifiers when application closes. On resume, the world's init method runs through the array and instantiates all dynamic objects (Appendix L). The objects then decode their own data.

4.2. Testing: Archiving of Dynamic Objects

A dynamic selectable object was added into the world and saving of object's position, rotation and movement target were tested. In the implementation from Section 2 the position and rotation of units were saved by the Motor module. However, when non-movable dynamic object was added the two variables needed to be saved by the Memory module since motor is a Model only available to units. The Memory module became responsible for saving of Controller's dynamic instance variables (Appendix E).
After the changes were made to the programme, it was able to archive all instantiated dynamic objects and resume them when an application started again. Moreover, if a unit was moving towards a target when the application closed, it resumed the movement when the programme was opened again.

4.3. Testing: Adding the DNA module

The final implemented function of units was breeding. Some functionality (especially the HandleTouchesMovedAt: method) needed to be added to UnitView (Appendix G) so that it could accept dragging. When a selected unit is dragged onto another unit, it is understood as the breeding command and the UserOrder: atLocation: withObject: method of DynObjController is called. The withObject: parameter is set as the target unit. Because of this parameter, a unit follows the mate before breeding happens.

A new Model UnitDNA was added as well. Debugger errors helped to create a list of rules for DynObjModel's subclasses listed in DynObjModel.h (Appendix D). After these rules were followed, UnitDNA was able to store and retrieve a DNA string and provide body and eyes colour to UnitController. Also, a function which mixes parent's DNA String was created. (Appendix I).
It became apparent that because units were added dynamically, the World needed to remember not only the number of units but also their array positions (since decoding needs a Model identifier and Controller's array position to properly identify an object). Therefore, the archivable array of object types mentioned in section 4.2. was implemented.

After the last changes were done, units were able to move and breed with others when dragged onto them and newly created units were successfully archived. Children had body and eyes colour independently and randomly inherited from one of the parents.

5. Conclusion

This work was aimed on creating a template project for a 2D iPhone game. Objectives of the template project were fast rendering, extendibility and persistence. The application created for the project implements a top-view world with static and dynamic objects and units that can move and breed. It could be extended for practically any iPhone game using any kind of 2D perspective. A rendering technique that draws only necessary objects and reuses a single texture for different images was created. The technique saves the device memory and computation time.

Behaviour-oriented approach of the subsumption architecture and the Model-View-Controller design pattern were used to deliver extendibility of the programme. Individual behavioural

modules of dynamic objects are independent of each other and of the application. All rendering and interaction with the user is a role of Views and independent of the Models. As a result of this functional separation, an efficient method of archiving of dynamic objects could be implemented. A general Model class provides the archiving methods and lets its subclasses overwrite them with their instance variables and class. Final testing proved that minimal work in terms of archiving needs to be done when additional Models are added.

The work showed that software development can benefit from the subsumption architecture borrowed from robotics. The created template project is fully extendible and modifiable thanks to the decorator and factory patterns which were used throughout the code.

The project can be downloaded from http://www.lenkaspace.net/downloads/2DiPhoneGameFramework.zip and used within Xcode.

References

Apple: MVC [online]
http://developer.apple.com/iphone/library/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html [accessed 06 January 2010]

Apple: Models [online]
http://developer.apple.com/iphone/library/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html [accessed 11 January 2010]

Apple: Archiving [online]
http://developer.apple.com/iPhone/library/docsumentation/General/Conceptual/DevPedia-CocoaCore/Archiving.html [accessed 11 January 2010]

Bourg D. M., Seeman G. (2004) *AI for Game Developers*. 1st ed. Sebastopol, O' Reilly Media.

Brooks, R.A. (1991). 'Intelligence without reason'. *Computers and Thought* IJCAI-91, April 1991

Fothergill A. (2009) 'Rapid Game Development Using (Mostly) Standard C'. *iPhone Games Projects*. New York, Apress

Franceschini N., Pichon J.M., Blanes C. (1992) From insect vision to Robot Vision. *Philosophical Transactions: Biological Sciences*, vol 337, p 283 - 294

Horswill I. D. and Brooks R. A. (1988) 'Situated vision in a dynamic world: Chasing objects'. *AAAI-88*, p. 796-800

Mackworth A.K. (1992) 'On Seeing Robots'. Computer Vision: Systems, Theory, and Applications p 1-13

Mark D. and LaMarche J. (2009) *Beginning iPhone Development*. New York, Apress

OODesign: Factory [online] http://www.oodesign.com/factory-pattern.html [accessed 10 January 2010]

OODesign: Decorator [online] http://www.oodesign.com/decorator-pattern.html [accessed 10 January 2010]

Webb B. (1996) 'A Cricket Robot'. *Scientific American*, December 1996, p 62-67

Appendix A: Texture2D

```
/*  File: Texture2D.h
 modified and simplified from Apple's sample code. */

#import <UIKit/UIKit.h>
#import <OpenGLES/ES1/gl.h>
#import "Constants.h"

extern int gNumSprites;
extern int gNumTextures;

typedef enum {
	kTexture2DPixelFormat_Automatic = 0,
	kTexture2DPixelFormat_RGBA8888,
	kTexture2DPixelFormat_RGB565,
	kTexture2DPixelFormat_A8,
} Texture2DPixelFormat;

typedef struct {
	GLuint name;
	GLfloat verts[12];
	GLfloat tverts[8];
} sprite_type;

typedef struct {
	GLuint		name;
	CGSize		size;
	NSUInteger	width,height;
	Texture2DPixelFormat format;
	GLfloat		maxS,maxT;

} texture_type;

void LoadTextureImage(CFStringRef filename,CFStringRef type,int sprname);
CGImageRef CreateNamedImage(CFStringRef imageName, CFStringRef type);
texture_type InitTexture(const void* data, Texture2DPixelFormat pixelFormat, NSUInteger width, NSUInteger
height, CGSize size, int sprname);

void CreateSprite(int a,float width,float height,float u1,float v1,float u2,float v2,GLuint text, int sprname);
void GrabSpriteSet(int across, int down, int count, int sprname);
void DrawSpriteAt(int a, CGPoint pos, CGFloat rot);
void DrawSpriteScaledAt(int a, CGPoint pos, CGPoint scale);
void DrawSpriteAtWithUV(int a,CGPoint pos, CGFloat rot, CGPoint uCoordins, CGPoint vCoordins);

BOOL g_bSelectedUnits[MAX_DYN_OBJS];
BOOL GetIsAnotherObjectSelected (int id_);

/*

 File: Texture2D.m
 Abstract: Creates OpenGL 2D textures from images or text.
  Originally based on Apple sample code from Crash Lander but trimmed, optimised, simplified and turned into
```

```
straight C
  */
#import <OpenGLES/ES1/glext.h>
#import "Texture2D.h"
#import "CreaturesAppDelegate.h"

//CONSTANTS:
#define kMaxTextureSize    1024

int spr[64];
sprite_type gSprite[64];
texture_type gTexture[64];
int gNumSprites = 0;
int gNumTextures = 0;
int gLastName;

BOOL GetIsAnotherObjectSelected (int id_){
      BOOL is = false;
      for (int i=0;i<MAX_DYN_OBJS; i++) {
          if (g_bSelectedUnits[i] == true && i!= id_) {
             is = true;
          }
      }
      return is;

}

void GrabSpriteSet(int across, int down, int count, int sprname )
{
      int b;
      texture_type lasttext;
      int x,y;
      GLfloat vertices[12];
      GLfloat coordinates[8];
      float width,height;
      GLfloat u1,u2,v1,v2;

      lasttext = gTexture[gNumTextures - 1];
      width = lasttext.width / across;
      height = lasttext.height / down;

      vertices[0] = -width / 2;              vertices[1] = -height / 2;     vertices[2] = 0;
      vertices[3] = width / 2;  vertices[4] = -height / 2;     vertices[5] = 0;
      vertices[6] = -width / 2;vertices[7] = height / 2;       vertices[8] = 0;
      vertices[9] = width / 2;  vertices[10] = height / 2;     vertices[11] = 0;

      for(y = 0; y < down && count > 0 ;y++) {
          for(x = 0; x < across && count > 0;x++) {
             spr[sprname++] = gNumSprites;
             u1 = (lasttext.maxS * x) / across;
             u2 = (lasttext.maxS * (x + 1)) / across;
             v1 = (lasttext.maxT * y) / down;
             v2 = (lasttext.maxT * (y + 1)) / down;
             coordinates[0] = u1; coordinates[1] = v2;
             coordinates[2] = u2; coordinates[3] = v2;
```

```
            coordinates[4] = u1; coordinates[5] = v1;
            coordinates[6] = u2; coordinates[7] = v1;

            gSprite[gNumSprites].name = lasttext.name;
            for(b = 0; b < 8;b++)
            {
                    gSprite[gNumSprites].tverts[b] = coordinates[b];
            }
            for(b = 0; b < 12;b++)
            {
                    gSprite[gNumSprites].verts[b] = vertices[b];
            }

            gNumSprites++;
            count--;

        }
    }

}

void DrawSpriteAt(int a,CGPoint pos, CGFloat rot)
{
    glPushMatrix();
    glTranslatef(pos.x,pos.y,0);
    glRotatef(rot, 0, 0, 1);
    if(gSprite[spr[a]].name == 0)
    {
        printf("spr %d\n",a);
    }
    else
    {
        glBindTexture(GL_TEXTURE_2D,gSprite[spr[a]].name);
        glVertexPointer(3,GL_FLOAT,0,gSprite[spr[a]].verts);
        glTexCoordPointer(2,GL_FLOAT,0,gSprite[spr[a]].tverts);
        glDrawArrays(GL_TRIANGLE_STRIP,0,4);
    }
    glPopMatrix();
}
/*  drawing method which uses part of a texture only  */
void DrawSpriteAtWithUV(int a,CGPoint pos, CGFloat rot, CGPoint uCoordins, CGPoint vCoordins)
{
    glPushMatrix();
    glTranslatef(pos.x,pos.y,0);
    glRotatef(rot, 0, 0, 1);
    if(gSprite[spr[a]].name == 0)
    {
        printf("spr %d\n",a);
    }
    else
    {
        /* calculate how the texture is sliced and which slice this is based on U coordins.
         Assuming that uCoordins.x < uCoordins.y
         */
        float uStep = uCoordins.y - uCoordins.x;
```

```
        float uPos = uCoordins.x / uStep;
        float vStep = vCoordins.y - vCoordins.x;
        float vPos = vCoordins.x / vStep;
        GLfloat textCoords[8];
        GLfloat textVerts[12];


        for (int i=0;i<8;i++) { // the array of UV cooridnates has 8 elements which represent 4 corners of the texture
          //------------ copy UV coordinates from the original texture:
          textCoords[i] = gSprite[spr[a]].tverts[i];
          if ( (i)%2 == 0) {
                textCoords[i] = textCoords[i]*uStep + uPos*uStep*gSprite[spr[a]].tverts[2]; // U coordinates
          } else {
                textCoords[i] = (vStep*textCoords[i]) + vPos*(vStep*gSprite[spr[a]].tverts[1]); // V coordinates
          }
        }

        for (int i=0;i<12;i++) { // the arra of XYZ texture coordinates has 12 elements which represent 4 corners in
3D.
          //---------------- copy dimensions of the original texture
          textVerts[i] = gSprite[spr[a]].verts[i];
          if ( (i)%3 == 0) {
                textVerts[i] *= uStep; // width (X) coordinates
          } else if ((i-1)%3 == 0) {
                textVerts[i] *= vStep; // height (Y) coordinates

          }
        }

        glBindTexture(GL_TEXTURE_2D,gSprite[spr[a]].name);
        glVertexPointer(3,GL_FLOAT,0,textVerts);
        glTexCoordPointer(2,GL_FLOAT,0,textCoords);
        glDrawArrays(GL_TRIANGLE_STRIP,0,4);
    }
    glPopMatrix();

}
```

Appendix B: DynObjController

```
#import <Foundation/Foundation.h>
#import "Sprite.h"

@class DynObjView;
@class DynObjMemory;
@class World;
@class DynObjModel;

@interface DynObjController : Sprite {
	DynObjView* m_pView;
	World* m_pWorld;
	Byte m_iType;
	NSInteger m_iToIdleCount;
	DynObjMemory* m_pMemory;
}

@property (nonatomic, retain) World* m_pWorld;
@property Byte m_iType;

- (id) initDynObj: (Byte)id_ type:(Byte)type_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_;
- (void) dealloc;
- (void) Update;
- (void) DoRendering;

- (BOOL) GetIsMoving;
- (void) UserOrder:(Byte)order_ atLocation:(CGPoint)loc_ withObject:(DynObjController*)withObj_;
- (void) Unselect;
- (void) GoIdle;
- (void) ScheduleIdleStateIn:(NSInteger)units_;
- (void) FinishedTask:(Byte)task_;

- (void) HandleTouchesBeginAt:(CGPoint)loc_;
- (void) HandleTouchesMovedAt:(CGPoint)loc_;
- (void) HandleTouchesEndedAt:(CGPoint)loc_;

- (NSString *) dataFilePathForKey: (NSString*)key_;
- (void) DecodeModelObject:(DynObjModel*) modelObject_ usingKey:(NSString*) objectKey_;
- (void) EncodeModelObject:(DynObjModel*) modelObject_ usingKey:(NSString*) objectKey_;

@end


#import "DynObjController.h"
#import "DynObjView.h"
#import "DynObjMemory.h"
#import "World.h"
#import "DynObjModel.h"

---

@implementation DynObjController

@synthesize m_pWorld;
```

```
@synthesize m_iType;

- (id) initDynObj: (Byte)id_ type:(Byte)type_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_{
    [super initStandard:id_ pos:pos_ size:CGPointMake(40,40) rot:rot_ txtrId:TEX_OBJS_DYN];
    m_pWorld = world_;
    m_iType = type_;
    m_pView = [[DynObjView alloc] initViewWithController:self];
    m_pMemory = [[DynObjMemory alloc] initMemoryWithController:self ];
    m_iToIdleCount = -1;
    m_ptUcoordins = CGPointMake(0,0.5);
    m_ptVcoordins = CGPointMake(0,1);
    return self;
}


- (void) dealloc {
    [m_pView release];
    [m_pMemory release];
    [super dealloc];
}
- (void) Update {
    //---------- determines whether to render:
    [super Update];
    //---------- test to idle count, descent it if scheduled for idle state and go idle when count reaches 0:
    if (m_iToIdleCount >= 0) {
        m_iToIdleCount --;
        if (m_iToIdleCount < 0) {
            [self GoIdle];
        }
    }
}

- (void) DoRendering {
    [m_pView RenderAt:m_ptPos rot:m_fRot txtrId:m_iTextureId uCoordins:m_ptUcoordins
vCoordins:m_ptVcoordins];
}

#pragma mark ================================ INTERFACE FOR MODEL CLASSES

- (BOOL) GetIsMoving {
    if ([m_pMemory GetCurTaskName] == UORDER_MOVE) {
        return true;
    } else {
        return false;
    }
}


#pragma mark ================================ USER INTERACTION

- (void) GoIdle{
    if ([m_pMemory GetCurTaskName] != UORDER_MOVE) {
        [self UserOrder:UORDER_NONE atLocation:CGPointMake(0,0) withObject:NULL];
        [self Unselect];
    }
```

```
}
- (void) ScheduleIdleStateIn:(NSInteger)units_ {
    m_iToIdleCount = units_;
}

- (void) UserOrder:(Byte)order_ atLocation:(CGPoint)loc_ withObject:(DynObjController*)withObj_ {
    [m_pMemory SetTaskTo:order_ atLocation:loc_ withObject:withObj_];
}

- (void) FinishedTask:(Byte)task_ {
    [self ScheduleIdleStateIn:SECONDS_UNTIL_IDLE*FPS];
    [m_pMemory SetTaskTo:UORDER_NONE atLocation:CGPointMake(0,0) withObject:NULL];
}

- (void) Unselect {
    g_bSelectedUnits[m_iId] = false;
    m_pView.m_bSelected = false;
}


#pragma mark ================================ TOUCHES
- (void) HandleTouchesBeginAt:(CGPoint)pos_ {
    [m_pView HandleTouchesBeginAt:pos_ selfPos:m_ptPos selfSize:m_ptSize];
}

- (void) HandleTouchesMovedAt:(CGPoint)pos_ {
    [m_pView HandleTouchesMovedAt:pos_ selfPos:m_ptPos selfSize:m_ptSize];
}

- (void) HandleTouchesEndedAt:(CGPoint)loc_ {
    [m_pView HandleTouchesEndedAt:loc_ selfPos:m_ptPos selfSize:m_ptSize];

}

#pragma mark ================================ ENCODING
/*
 The function returns path to a data file for a specified object key in form objectKey_controllerId.
 The '_controllerId' part of the file path makes sure no two objects get encoded into the same file.
 */
- (NSString *) dataFilePathForKey: (NSString*)key_ {
    //--- look for the Documents directory within the application's sandbox
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask,
YES);
    NSString *documentsDir = [paths objectAtIndex:0];
    NSString *fileName =[[NSString alloc] initWithFormat:@"%@_%d",key_,m_iId];
    //------- append the filename to get its full path:
    return [documentsDir  stringByAppendingPathComponent:fileName];
    [fileName release];
    [paths release];
    [documentsDir release];
}

- (void) DecodeModelObject:(DynObjModel*) modelObject_ usingKey:(NSString*) objectKey_{
    NSString *filePath = [self dataFilePathForKey:objectKey_];
    //------------ if file exists, copy its content into data object and create a new instance:
```

```
    if ([[NSFileManager defaultManager] fileExistsAtPath:filePath]) {
        NSData *data = [[NSMutableData alloc] initWithContentsOfFile:[self dataFilePathForKey:objectKey_]];
        NSKeyedUnarchiver *unarchiver = [[NSKeyedUnarchiver alloc] initForReadingWithData:data];
        DynObjModel * Model = [unarchiver decodeObjectForKey:objectKey_];
        [modelObject_ SetSelfWith:Model];
        [unarchiver finishDecoding];
        [unarchiver release];
        [data release];
    }
}

- (void)EncodeModelObject:(DynObjModel*)modelObject_ usingKey:(NSString*)objectKey_ {
    NSMutableData *data = [[NSMutableData alloc] init];
    NSKeyedArchiver *archiver = [[NSKeyedArchiver alloc] initForWritingWithMutableData:data];
    [archiver encodeObject:modelObject_ forKey: objectKey_];
    [archiver finishEncoding];
    [data writeToFile:[self dataFilePathForKey:objectKey_] atomically:YES];
    //---------
    [archiver release];
    [data release];

}


@end
```

Appendix C: DynObjView

```
#import <Foundation/Foundation.h>
#import "Texture2D.h"


@class DynObjController;

@interface DynObjView : NSObject {
	BOOL m_bSelected;
	DynObjController *m_pController;
	BOOL m_bGoingToUnselect;
}

@property  BOOL m_bSelected;

- (id) initViewWithController: (DynObjController*)control_;
- (void) RenderAt: (CGPoint)pos_ rot:(CGFloat)rot_ txtrId: (Byte)txtrId_ uCoordins:(CGPoint)uc_
vCoordins:(CGPoint)vc_;
- (BOOL) HitTest:(CGPoint)withPt selfPos:(CGPoint)pos_ selfSize:(CGPoint)size_;

- (void) HandleTouchesBeginAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_;
- (void) HandleTouchesMovedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_;
- (void) HandleTouchesEndedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_;


- (void) UserStartTouchInside;
- (void) UserStartTouchOutside;

@end
```

```
#import "DynObjView.h"
#import "DynObjController.h"
#import "Texture2D.h"
#import "World.h"

@implementation DynObjView

@synthesize m_bSelected;

- (id) initViewWithController: (DynObjController*)control_ {
	[super init];
	m_pController = control_;
	m_bSelected = false;
	return self;
}
- (void) RenderAt: (CGPoint)pos_ rot:(CGFloat)rot_ txtrId:(Byte)txtrId_ uCoordins:(CGPoint)uc_
vCoordins:(CGPoint)vc_ {
	CGPoint uCoords = uc_;
	//---------- adjust U texture coordinates if selected, use slice on the right:
	if (m_bSelected) {
	    if (![m_pController GetIsMoving]) {
```

```
            uCoords = CGPointMake(uc_.x+0.5,uc_.y + 0.5);
        }

    }
    DrawSpriteAtWithUV(txtrId_,pos_,rot_, uCoords, vc_);
}
/* HitTest
 - returns true if point is inside of the view
*/
- (BOOL) HitTest: (CGPoint)withPt selfPos:(CGPoint)pos_ selfSize:(CGPoint)size_ {
    BOOL hit = false;
    if (withPt.y > pos_.y - size_.y/2 && withPt.y < pos_.y + size_.y/2) {
        if (withPt.x > pos_.x - size_.x/2 && withPt.x < pos_.x + size_.x/2) {
            hit = true;
        }
    }
    return hit;
}

#pragma mark ================================ USER INTERACTION

- (void) HandleTouchesBeginAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_ {
    //-------------------- test if object has been clicked on:
    if ([self HitTest:pos_ selfPos:selfPos_ selfSize:selfSize_]) {
        [self UserStartTouchInside];
    } else {
        //---------------- object hasn't been clicke on:
        [self UserStartTouchOutside];

    }
    //---------------- register self in the global array as selected:
    if (m_bSelected) {
        g_bSelectedUnits[m_pController.m_iId] = true;
    }

}

- (void) UserStartTouchInside {
    [m_pController.m_pWorld UnselectAllOtherObjects:m_pController.m_iId];
    m_bSelected = !m_bSelected;

}
- (void) UserStartTouchOutside { }

- (void) HandleTouchesMovedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_{
}

- (void) HandleTouchesEndedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_{
}

@end
```

Appendix D: DynObjModel

```
/*  DynObjModel

 - base class for all dynamic object's models
 - a subclass should implement the following:
      1. Custom init function (eg. - (id) initMyModelWithController: (DynObjController*)control_;)
      2. In the init function AFTER INITIALISING CLASS VARIABLES call [super initModelWithController: encodingKey:] The encoding key corresponds to a name under which a model is encoded and should be unique for each class.
      3. Encoding / decoding methods:
         A) - (void) encodeWithCoder: (NSCoder*) encoder
         B) - (id) initWithCoder: (NSCoder *) decoder
         C) - (void) SetSelfWith: (DynObjModel*)decodedObj_
      4. Make all encoded variables properties of the class

      Further details about these functios in DynObjModel.m
*/

#import <Foundation/Foundation.h>
#import "Constants.h"

#define k1 @"var1"
#define k2 @"var2"
#define k3 @"var3"
#define k4 @"var4"
#define k5 @"var5"
#define k6 @"var6"
#define k7 @"var7"
#define k8 @"var8"
#define k9 @"var9"
#define k10 @"var10"

@class DynObjController;

@interface DynObjModel : NSObject <NSCoding> {
    DynObjController* m_pController;
    NSString * m_sEncodingKey;
}

@property(nonatomic, retain) DynObjController *m_pController;

- (id) initModelWithController:(DynObjController*)controller_ encodingKey:(NSString*)key_;
- (void) SetSelfWith: (DynObjModel*)decodedObj_;

@end
```

```
#import "DynObjModel.h"
#import "DynObjController.h"
@implementation DynObjModel

@synthesize m_pController;
```

```
- (id) initModelWithController:(DynObjController*)controller_ encodingKey:(NSString*)key_{
    [super init];
    m_pController = controller_;
    m_sEncodingKey = key_;
    //----------------- call AgentController's universal method for Model decoding. AgentController determines whether encoded data for specific class exist.
    [m_pController DecodeModelObject: self usingKey: key_];

    //--------- get reference to the application instance
    UIApplication *app = [UIApplication sharedApplication];

    //---------- subscribe to the application terminate notification, using this class as observer (implements a method that handles it specified as selector. Name of the notification is a keyword. Object specifies which object notifies.
[[NSNotificationCenter defaultCenter] addObserver:self
    selector:@selector(applicationWillTerminate:)
name:UIApplicationWillTerminateNotification              object:app];
    return self;
}

- (void) applicationWillTerminate: (NSNotification *) notification {
    [m_pController EncodeModelObject:self usingKey:m_sEncodingKey];
}

#pragma mark ===================================== NSCoding
/*
 Encode and decode class variables of individual Model objects. Only variables which change their value during the program runs need to be encoded/decoded. */

/*
 encodeWithCoder:
 - encodes an object when application closes
*/
- (void) encodeWithCoder: (NSCoder*) encoder {
    /* to encode a varialbe use e.g:
     [encoder encodeInteger: m_iMyInt forKey: k1];
     - individual keys are supplied in the header of this class.
     */
}

/* initWithCoder:
 - decodes and object when application starts
*/
- (id) initWithCoder: (NSCoder *) decoder {

    if (self == [super init]) {
        /* decode variables listed in encodeWithCoder: e.g.
         m_iMyInt = [decoder decodeIntegerForKey:k1];
         */
    }
    return self;
}

/* SetSelfWith:
```

```
 - copies properties over from a decoded object
 - !!! set the class of decodedObj_ to the class which implements the method
 */
- (void) SetSelfWith: (DynObjModel*)decodedObj_ {
      /* copy properties listed in encodeWithCoder: e.g
      m_iMyInt = decodedObj_.m_iMyInt;
       */
}
@end
```

Appendix E: DynObjMemory

```
#import <Foundation/Foundation.h>
#import "DynObjModel.h"

typedef struct {
	Byte name;
	CGPoint loc;
	DynObjController* targetObj;
} Task;

@class DynObjController;

@interface DynObjMemory : DynObjModel{
	Task m_CurTask;
	CGPoint m_ptPos;
	CGFloat m_fRot;
}

@property Task m_CurTask;
@property CGPoint m_ptPos;
@property CGFloat m_fRot;

- (id) initMemoryWithController:(DynObjController*)control_ ;
- (void) Update;
- (void) SetTaskTo:(Byte)name_ atLocation:(CGPoint)loc_ withObject:(DynObjController*)withObj_;
- (Byte) GetCurTaskName;
- (CGPoint) GetCurTaskLocation;
- (DynObjController*) GetCurTaskTargetObj;
- (void) SetSelfWith: (DynObjMemory*)decodedObj_;

@end
```

```
#import "DynObjMemory.h"
#import "DynObjController.h"

@implementation DynObjMemory

@synthesize m_CurTask;
@synthesize m_ptPos;
@synthesize m_fRot;

- (id) initMemoryWithController:(DynObjController*)control_ {
	m_pController = control_;
	m_CurTask.name = UORDER_NONE;
	m_CurTask.targetObj = NULL;
	[super initModelWithController:control_ encodingKey:@"dataMemory"];
	return self;
}
- (void) Update{
	BOOL finished = false;
	//----------------- decide if task finished yet:
```

```
    switch (m_CurTask.name) {
        case UORDER_BREED: {
            NSInteger tolerance = 50;
            if (m_pController.m_ptPos.x <= m_CurTask.targetObj.m_ptPos.x + tolerance &&
m_pController.m_ptPos.x >= m_CurTask.targetObj.m_ptPos.x - tolerance) {
                if (m_pController.m_ptPos.y <= m_CurTask.targetObj.m_ptPos.y + tolerance &&
m_pController.m_ptPos.y >= m_CurTask.targetObj.m_ptPos.y - tolerance) {
                    finished = true;
                }
            }
        } break;
        case UORDER_MOVE: {
            NSInteger tolerance = 25;
            if (m_pController.m_ptPos.x <= m_CurTask.loc.x + tolerance && m_pController.m_ptPos.x >=
m_CurTask.loc.x - tolerance) {
                if (m_pController.m_ptPos.y <= m_CurTask.loc.y + tolerance && m_pController.m_ptPos.y >=
m_CurTask.loc.y - tolerance) {
                    finished = true;
                }
            }
        } break;
    }
    //----------------- if task finished set current task to none:
    if (finished) {
        [m_pController FinishedTask:m_CurTask.name];
    }
}

- (void) SetTaskTo:(Byte)name_ atLocation:(CGPoint)loc_ withObject:(DynObjController*)withObj_{
    m_CurTask.name = name_;     m_CurTask.loc = loc_;       m_CurTask.targetObj = withObj_;
}

- (Byte) GetCurTaskName {           return m_CurTask.name; }
- (CGPoint) GetCurTaskLocation { return m_CurTask.loc; }
- (DynObjController*) GetCurTaskTargetObj { return m_CurTask.targetObj; }

#pragma mark ===================================== NSCoding
- (void) encodeWithCoder: (NSCoder*) encoder {
    //---------- encode each property and DynObjController's properties:
    [encoder encodeInteger: m_CurTask.name forKey: k1];
    [encoder encodeCGPoint:m_CurTask.loc forKey: k2];
    [encoder encodeCGPoint:m_pController.m_ptPos forKey: k3];
    [encoder encodeFloat:m_pController.m_fRot forKey: k4];
}

//---------------- decode:
- (id) initWithCoder: (NSCoder *) decoder {
    if (self == [super init]) {
        m_CurTask.name = [decoder decodeIntegerForKey:k1];
        m_CurTask.loc= [decoder decodeCGPointForKey:k2];
        m_ptPos = [decoder decodeCGPointForKey:k3];
        m_fRot = [decoder decodeFloatForKey:k4];
    }
    return self;
}
```

```
- (void) SetSelfWith: (DynObjMemory*)decodedObj_ {
	m_CurTask.name = decodedObj_.m_CurTask.name;
	m_CurTask.loc = decodedObj_.m_CurTask.loc;
	m_pController.m_ptPos = decodedObj_.m_ptPos;
	m_pController.m_fRot = decodedObj_.m_fRot;
}
@end
```

Appendix F: UnitController

```
#import <Foundation/Foundation.h>
#import "DynObjController.h"

@class UnitMotor;
@class UnitDNA;

@interface UnitController : DynObjController {
	UnitMotor* m_pMotor;
	UnitDNA* m_pDNA;
}

- (id) initUnit: (Byte)id_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_;
- (id) initUnit: (Byte)id_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_ DNA:dna_;
- (void) SetupSelf;
- (void) dealloc;
- (void) Update;

- (void) FinishedTask:(Byte)task_;
- (NSString*) GetDNA;

@end


/*  REFERENCES
 [1]  http://stackoverflow.com/questions/1230289/override-classes-variable-or-at-least-variable-type-in-objective-c-cocoa (accessed 10 Jan 2010)
*/
#import "UnitController.h"

#import "UnitMotor.h"
#import "World.h"
#import "DynObjMemory.h"
#import "UnitView.h"
#import "UnitDNA.h"

@implementation UnitController

/* CONSTRUCTOR 1
 - used from World class when starting the program
*/
- (id) initUnit: (Byte)id_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_{
	[super initDynObj:id_ type:OBJ_UNIT pos:pos_ rot:rot_ world:world_];
	m_pDNA = [[UnitDNA alloc] initDNAWithController:self];
	[self SetupSelf];
	return self;
}

/* CONSTRUCTOR 2
 - used from World class when a unit is added after breeding using a specific DNA string
*/
- (id) initUnit: (Byte)id_ pos:(CGPoint)pos_ rot:(CGFloat)rot_ world:(World*)world_ DNA:(NSString*)dna_{
```

```
    [super initDynObj:id_ type:OBJ_UNIT pos:pos_ rot:rot_ world:world_];
    m_pDNA = [[UnitDNA alloc] initDNAWithController:self];
    m_pDNA.m_sDNAString = dna_;
    [self SetupSelf];
    return self;
}

- (void) SetupSelf {
    //------ change texture:
    m_iTextureId = TEX_UNITS_BODIES;
    m_pMotor = [[UnitMotor alloc] initMotorWithController:self];
    m_pView = [[UnitView alloc] initViewWithController:self];
    //----------- determine slice of the units texture:
    m_ptUcoordins = CGPointMake(0,0.5);
    Byte bodyCol = [m_pDNA GetBodyColor];
    switch (bodyCol) {
        case COL_PURPLE:        m_ptVcoordins = CGPointMake(0,0.25); break;
        case COL_RED: m_ptVcoordins = CGPointMake(0.25,0.5); break;
        case COL_BLUE: m_ptVcoordins = CGPointMake(0.5,0.75); break;
        case COL_GREEN: m_ptVcoordins = CGPointMake(0.75,1); break;
    }
}

- (void) dealloc {
    [m_pMotor release];
    [m_pDNA dealloc];
    [m_pDNA release];
    [super dealloc];
}
- (void) Update {
    [super Update];
    [m_pMemory Update];
    //--------------------- decide what to do based on the current task:
    if ([m_pMemory GetCurTaskName] == UORDER_MOVE ) {
        [m_pMotor MoveTowards:[m_pMemory GetCurTaskLocation]];
    } else if ([m_pMemory GetCurTaskName] == UORDER_BREED) {
        CGPoint pos = [m_pMemory GetCurTaskTargetObj].m_ptPos;
        [m_pMotor MoveTowards:pos];
    }
}

- (void) DoRendering {
    //----------- call UnitView's overriden RenderAt function  [REF 1]:
    [(UnitView*)m_pView RenderAt:m_ptPos rot:m_fRot txtrId:m_iTextureId uCoordins:m_ptUcoordins
vCoordins:m_ptVcoordins eyes:[m_pDNA GetEyesColor]];
}

- (NSString*) GetDNA {    return m_pDNA.m_sDNAString; }

#pragma mark ================================ USER INTERACTION

- (void) FinishedTask:(Byte)task_  {
    switch (task_) {
        case UORDER_BREED: {
          NSString*    newDNA    =    [m_pDNA    GenerateChildStringWith:[(UnitController*)[m_pMemory
```

```
GetCurTaskTargetObj] GetDNA]];
            [m_pWorld AddUnitAt:CGPointMake(m_ptPos.x, m_ptPos.y) rot:0 withDNA:newDNA];
        }
            break;
    }
    [super FinishedTask: task_];
}

@end
```

Appendix G: UnitView

```
#import <Foundation/Foundation.h>
#import "DynObjView.h"

@class UnitController;

@interface UnitView : DynObjView {
}

- (id) initViewWithController: (UnitController*)control_;
- (void) RenderAt: (CGPoint)pos_ rot:(CGFloat)rot_ txtrId: (Byte)txtrId_ uCoordins:(CGPoint)uc_
vCoordins:(CGPoint)vc_ eyes:(Byte)eyesCol_;
- (void) UserStartTouchInside;
- (void) HandleTouchesMovedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_;
- (void) HandleTouchesEndedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_;

@end
```

```
#import "UnitView.h"
#import "UnitController.h"
#import "Vector.h"
#import "World.h"


@implementation UnitView

- (id) initViewWithController: (UnitController*)control_ {
    [super initViewWithController:control_];
    return self;
}

- (void) RenderAt: (CGPoint)pos_ rot:(CGFloat)rot_ txtrId: (Byte)txtrId_ uCoordins:(CGPoint)uc_
vCoordins:(CGPoint)vc_ eyes:(Byte)eyesCol_ {
    //-------- draw body based on coordinates set in constructor
    [super RenderAt:pos_ rot:rot_ txtrId:txtrId_ uCoordins:uc_ vCoordins:vc_];
    //--------- render eyes:
    CGPoint eyesU = CGPointMake(0,0.5);
    CGPoint eyesV;
    switch(eyesCol_) {
        case COL_GREEN:
            eyesV = CGPointMake(0,0.25);
            break;
        case COL_BROWN:
            eyesV = CGPointMake(0.25,0.5);
            break;
        case COL_PURPLE:
            eyesV = CGPointMake(0.5,0.75);
            break;
        case COL_ORANGE:
            eyesV = CGPointMake(0.75,1);
```

```
            break;
        }

        CGPoint eyesPos;
        //------------------ decide about eyes' U coordinates and position
        if (m_bSelected && ![m_pController GetIsMoving]) {
            eyesU = CGPointMake(0.5,1);
            eyesPos= AddVectors(m_pController.m_ptPos, CGPointMake(

                -cos(m_pController.m_fRot*PI/180)*5,

                -sin(m_pController.m_fRot*PI/180)*5));
        } else {
            eyesPos= AddVectors(m_pController.m_ptPos, CGPointMake(

              cos(m_pController.m_fRot*PI/180)*25,

              sin(m_pController.m_fRot*PI/180)*25));
        }
        DrawSpriteAtWithUV(TEX_UNITS_EYES, eyesPos, rot_, eyesU, eyesV);

}

- (void) UserStartTouchInside {
        [super UserStartTouchInside];
        [m_pController UserOrder:UORDER_NONE atLocation:CGPointMake(0,0) withObject:NULL];
}

- (void) HandleTouchesMovedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_{
        if (!m_bSelected && g_bSelectedUnits[m_pController.m_iId] == true) {
            m_bSelected = true;
        }
}

- (void) HandleTouchesEndedAt:(CGPoint)pos_ selfPos:(CGPoint)selfPos_ selfSize:(CGPoint)selfSize_{
        if (m_bSelected && !GetIsAnotherObjectSelected(m_pController.m_iId)) {
            //----- test if touch ended on another unit, if yes multiply order
            int otherDynObjPos = [m_pController.m_pWorld GetDynObjIdAt:pos_];
            NSInteger type = [m_pController.m_pWorld GetDynObjWithId:otherDynObjPos].m_iType;
            if (otherDynObjPos > -1 &&  type == OBJ_UNIT && otherDynObjPos != m_pController.m_iId) {
                [m_pController  UserOrder:UORDER_BREED  atLocation:pos_  withObject:[m_pController.m_pWorld
GetDynObjWithId:otherDynObjPos]];
            } else {
            //---------- if not, move order
                [m_pController UserOrder:UORDER_MOVE atLocation:pos_ withObject:NULL];
            }
        }
}

@end
```

Appendix H: UnitMotor

```
#import <Foundation/Foundation.h>
#import "DynObjModel.h"

typedef struct {
    CGFloat steeringL;    CGFloat steeringR;        CGFloat thrustBack;        CGFloat thrustFront;
} Motors;

@class UnitController;

@interface UnitMotor : DynObjModel{
    CGFloat m_fSpeed;
    Motors m_sMotors;
    NSMutableDictionary *m_dicVarsList;
}

@property Motors m_sMotors;

- (id) initMotorWithController:(UnitController*)control_ ;
- (void) MoveTowards: (CGPoint)loc_;
- (void) GoForward;
- (void) TurnBy:(CGFloat) angle_;

@end
```

```
#import "UnitMotor.h"
#import "Vector.h"
#import "UnitController.h"

@implementation UnitMotor

@synthesize m_sMotors;

/*
 main init method, called each time application starts from the AgentController class
*/
- (id) initMotorWithController:(UnitController*)control_ {
    m_fSpeed = 10;
    m_sMotors.steeringL = m_sMotors.steeringR = 0;
    m_sMotors.thrustBack = m_sMotors.thrustFront = 0;
    //---------- the super class method which handles decoding and subscribes to applicationWillTermina
notification:
    [super initModelWithController:control_ encodingKey:@"dataMotor"];
    return self;
}

- (void) MoveTowards: (CGPoint)loc_ {
    CGPoint localVec = ConvertPointToLocalCoordins (m_pController.m_fRot,SubtractVectors(loc_,
m_pController.m_ptPos));
    //---------- adjust speed according to the distance of the target, minimal speed is 3:
    m_sMotors.thrustBack = 3 + fabs(localVec.y / 50);
```

```
    //------------ normalize the local vector to get only direction:
    localVec = NormalizeVector(localVec);
    //----------- use vector's x component to turn:
    [self TurnBy:localVec.x*5];
    //------------------ test if user clicked along the local horizontal axis:
    if (localVec.x < 0.1 && localVec.x > -0.1) {
        //----------- test if click was behind:
        if (localVec.y > 0) {
            //------- make a big turn:
            [self TurnBy:90];
        }
    }
    [self GoForward];

}

#pragma mark ================================== BASIC

- (void) GoForward {
    m_pController.m_ptPos = AddVectors(m_pController.m_ptPos, CGPointMake(
            cos(m_pController.m_fRot*PI/180)*m_sMotors.thrustBack,
            sin(m_pController.m_fRot*PI/180)*m_sMotors.thrustBack));
}

- (void) TurnBy:(CGFloat) angle_ {
    m_pController.m_fRot += angle_;
}


#pragma mark ====================================== NSCoding
/* no archiving neccessary, UnitMotor does not have any instance variables which need to persist */

@end
```

Appendix I: UnitDNA

```
#import <Foundation/Foundation.h>
#import "DynObjModel.h"

@class UnitController;

@interface UnitDNA : DynObjModel {
    NSString* m_sDNAString;
}

@property (nonatomic, retain) NSString * m_sDNAString;

- (id) initDNAWithController:(UnitController*) control_;
- (void) SetSelfWith: (UnitDNA*)decodedObj_;
- (Byte) GetBodyColor;
- (Byte) GetEyesColor;
- (NSString*) GenerateChildStringWith:(NSString*)otherDNA_;

@end
```

```
/* REFERENCES

 [1] Mark D. and LaMarche J. (2009) Beginning iPhone Development. New York, Apress, p. 173
 [2]  http://stackoverflow.com/questions/1518309/objective-c-how-do-i-substring-an-email-address-in-text-field-to-firstpart-separ (accessed 09 Jan 2010)
 */

#import "UnitDNA.h"
#import "UnitController.h"

@implementation UnitDNA

@synthesize m_sDNAString;

- (id) initDNAWithController:(UnitController*) control_ {
    //----- generate random body and eyes color:
    NSInteger bodyCol = random() % 4; // [REF 1]
    NSInteger eyesCol = random() % 4;
    m_sDNAString =  [[NSString alloc] initWithFormat:@"%d_%d",bodyCol,eyesCol];
    [super initModelWithController:control_ encodingKey:@"dataDNA"];
    return self;
}

- (void) dealloc {
    [m_sDNAString release];
    [super dealloc];
}

- (Byte) GetBodyColor {
    NSString* bodyColStr = [[m_sDNAString componentsSeparatedByString:@"_"] objectAtIndex:0]; //[REF 2]
    NSInteger num = [bodyColStr intValue];
```

```
    Byte returnVal;
    switch (num) {
        case 0: returnVal = COL_PURPLE; break;
        case 1: returnVal = COL_RED;    break;
        case 2: returnVal = COL_BLUE; break;
        case 3: returnVal = COL_GREEN; break;
    }
    return returnVal;
}

- (Byte) GetEyesColor {
    NSString* eyesColStr = [[m_sDNAString componentsSeparatedByString:@"_"] objectAtIndex:1]; //[REF 2]
    NSInteger num = [eyesColStr intValue];
    Byte returnVal;
    switch (num) {
        case 0: returnVal = COL_GREEN; break;
        case 1: returnVal = COL_BROWN; break;
        case 2: returnVal = COL_PURPLE; break;
        case 3: returnVal = COL_ORANGE; break;
    }
    return returnVal;
}

- (NSString*) GenerateChildStringWith:(NSString*)otherDNA_ {
    NSInteger temp = random() % 200;
    //---------------------- generate new body color from parent 1 or 2:
    NSInteger newBodyCol;
    if (temp <=100) {
        newBodyCol = [[[m_sDNAString componentsSeparatedByString:@"_"] objectAtIndex:0] intValue];
    } else {
        newBodyCol = [[[otherDNA_ componentsSeparatedByString:@"_"] objectAtIndex:0] intValue];
    }
    //---------------------- generate new eyes color from parent 1 or 2:
    NSInteger newEyesCol;
    temp = random() % 200;
    if (temp <= 100) {
        newEyesCol = [[[m_sDNAString componentsSeparatedByString:@"_"] objectAtIndex:1] intValue];
    } else {
        newEyesCol = [[[otherDNA_ componentsSeparatedByString:@"_"] objectAtIndex:1] intValue];
    }
    NSString* newDNA = [NSString stringWithFormat:@"%d_%d",newBodyCol,newEyesCol];
    return newDNA;
}

#pragma mark ===================================== NSCoding
/* details about the methods in DynObjModel.m */
- (void) encodeWithCoder: (NSCoder*) encoder {
     [encoder encodeObject: m_sDNAString forKey: k1];
}
- (id) initWithCoder: (NSCoder *) decoder {
    if (self == [super init]) {
        self.m_sDNAString = [decoder decodeObjectForKey:k1];
    }
    return self;
}
```

```
- (void) SetSelfWith: (UnitDNA*)decodedObj_  {
      self.m_sDNAString = decodedObj_.m_sDNAString;

}

@end
```

Appendix J: Contants.h

```
#define PI 3.1415926535897
#define SCR_W 240
#define SCR_H 160

#define MAX_STAT_OBJS 10
#define MAX_DYN_OBJS 30

#define FPS 60
#define SECONDS_UNTIL_IDLE 10

/*    TEXTURES
 */
enum {
    TEX_TERRAIN = 0,    TEX_UNITS_BODIES,    TEX_UNITS_EYES,        TEX_OBJS_STAT,
    TEX_OBJS_DYN,
};

/* USER ORDERS
    - used for communication of user commands between Agent classes
 */
enum {
    UORDER_MOVE = 0,            UORDER_STOP,            UORDER_NONE,            UORDER_BREED,
};


/* TYPES OF OBJECTS
 */

enum {
    OBJ_SMALL = 0,      OBJ_BIG,        OBJ_OPENINGHOLE,    OBJ_UNIT,
};

/*  TYPES OF COLORS
 */
enum {
    COL_PURPLE = 0,    COL_RED,        COL_BLUE,       COL_GREEN,    COL_ORANGE,
    COL_BROWN,
};
```

Appendix K: Vector

```
/*  provides static methods for manipulation with vectors, i.e. no instance necessary to use these methods */

#import <Foundation/Foundation.h>

CGPoint ConvertPointToLocalCoordins (CGFloat orientation_,CGPoint vector_);
CGPoint SubtractVectors(CGPoint vec1,CGPoint  vec2);
CGPoint AddVectors(CGPoint vec1, CGPoint vec2);
CGPoint NormalizeVector(CGPoint vec_);
```

```
/* REFERENCE:
Rewritten from C++: Bourg D. M., Seeman G. (2004) AI for Game Developers. 1st ed. Sebastopol, O' Reilly Media. */

#import <OpenGLES/ES1/glext.h>
#import "Vector.h"
#import "Constants.h"

CGPoint ConvertPointToLocalCoordins (CGFloat orientation_,CGPoint vector_){
    CGPoint tempVec;
    tempVec.x = vector_.x*cos((orientation_+90)*PI/180) + vector_.y*sin((orientation_+90)*PI/180);
    tempVec.y = - vector_.x*sin((orientation_+90)*PI/180) + vector_.y*cos((orientation_+90)*PI/180);
    return tempVec;
}
CGPoint SubtractVectors(CGPoint vec1,CGPoint  vec2) {
    CGPoint result;
    result.x=vec1.x-vec2.x;          result.y=vec1.y-vec2.y;
    return result;
}
CGPoint AddVectors(CGPoint vec1,CGPoint  vec2) {
    CGPoint result;
    result.x=vec1.x+vec2.x;          result.y=vec1.y+vec2.y;
    return result;
}
CGPoint NormalizeVector(CGPoint vec_) {
    CGPoint vec = vec_;
    float tolerance = 0.0001;
    float m = sqrt((vec.x* vec.x) + (vec.y*vec.y));
    if (m <= tolerance) {m=1;}
    vec.x /=m;
    vec.y /=m;

    if (fabs(vec.x) < tolerance){vec.x = 0;}
    if (fabs(vec.y) < tolerance){vec.y = 0;}

    return vec;
}
```

Appendix L: World

```
#import <Foundation/Foundation.h>
#import "Constants.h"
#define SETTINGS_FILE_NAME @"settings"

@class DynObjController;
@class StatObj;

@interface World : NSObject {
	StatObj *m_pStaticObjects [MAX_STAT_OBJS];
	DynObjController *m_pDynamicObjects [MAX_DYN_OBJS];
	NSInteger m_iNumStaticObjects;
	NSInteger m_iNumDynamicObjects;
	CGPoint m_ptGlobalCoordins;

}

- (id) initWorld;
- (void) dealloc;
- (void) DoFrame;
- (void) AddUnitAt: (CGPoint)pos_ rot:(CGFloat)rot_;
- (void) AddUnitAt: (CGPoint)pos_ rot:(CGFloat)rot_ withDNA:(NSString*)dna_;
- (void) AddStaticObjectAt: (CGPoint)pos_ rot:(CGFloat)rot_ type:(Byte)type_;
- (void) AddDynamicObjectAt: (CGPoint)pos_ rot:(CGFloat)rot_ type:(Byte)type_;
- (void) HandleTouchesBeginAt:(CGPoint)loc_;
- (void) HandleTouchesMovedAt:(CGPoint)loc_;
- (void) HandleTouchesEndedAt:(CGPoint)loc_;
- (DynObjController*) GetDynObjWithId:(NSInteger)id_;
- (void) UnselectAllOtherObjects:(NSUInteger)id_;
- (int) GetDynObjIdAt: (CGPoint)pos_ ;

- (NSString *) dataFilePath;
- (void) applicationWillTerminate: (NSNotification *) notification;

@end


/* REFERENCES
 [1]  http://www.iphonedevsdk.com/forum/iphone-sdk-development/4717-nsmutablearray-integers.html  (accessed
10 January 2010)   */

#import "World.h"
#import "UnitController.h"
#import "StatObj.h";
#import "DynObjController.h";

@implementation World

- (id) initWorld {
	[super init];
	m_ptGlobalCoordins.x = 0; m_ptGlobalCoordins.y = 0;
	//---------------------------------- rewrite program settings if they exist:
```

```
    NSString *filePath = [self dataFilePath];
    //------------- if file exists, copy its content into array and create dynamic objects:
    if ([[NSFileManager defaultManager] fileExistsAtPath:filePath]) {
        NSArray *array = [[NSArray alloc] initWithContentsOfFile:filePath];
        int num = [array count];
        for (int i=0;i<num;i++ ) {
            Byte objType = [[array objectAtIndex:i] intValue];
            switch (objType) {
                    case OBJ_UNIT:  [self AddUnitAt:CGPointMake(0,0) rot:0 ]; break;
                    case OBJ_OPENINGHOLE: [self AddDynamicObjectAt:CGPointMake(0,0) rot:0
type:OBJ_OPENINGHOLE]; break;
            }
        }
        [array release];
    } else {
        //-------------- when application runs for the first time:
        [self AddUnitAt:CGPointMake(-200,-120) rot:0 ];
        [self AddUnitAt:CGPointMake(-100,100) rot:90 ];
        [self AddUnitAt:CGPointMake(100,-150) rot:180 ];
        [self AddUnitAt:CGPointMake(200,50) rot:270 ];
        [self AddDynamicObjectAt:CGPointMake(150,150) rot:0 type:OBJ_OPENINGHOLE];
    }
    [self AddStaticObjectAt:CGPointMake(0,0) rot:0 type:OBJ_BIG];

    //--------- get reference to the application instance
    UIApplication *app = [UIApplication sharedApplication];
    //---------- subscribe to the notification, using this class as observer (implements a method that handles it
specified as selector. Name of the notification is a keyword. Object specifies which object notifies.
    [[NSNotificationCenter defaultCenter] addObserver:self  selector:@selector(applicationWillTerminate:
                name:UIApplicationWillTerminateNotification object:app];
    return self;

}

- (void) dealloc {
    for (NSInteger i=0; i< m_iNumStaticObjects; i++) {
        [m_pStaticObjects[i] dealloc];   [m_pStaticObjects[i] release];
    }
    [super dealloc];
}

- (void) DoFrame {
    //------------------------------------------------------ UPDATE WORLD
    for (NSInteger i=0; i< m_iNumStaticObjects; i++) {
        [m_pStaticObjects[i] Update];
    }
    for (NSInteger i=0; i< m_iNumDynamicObjects; i++) {
        [m_pDynamicObjects[i] Update];
    }
    //------------------------------------------------------ RENDER WORLD
    DrawSpriteAt(TEX_TERRAIN, m_ptGlobalCoordins,0); // draw terrain

    for (NSInteger i=0; i< m_iNumStaticObjects; i++) {
        [m_pStaticObjects[i] Render];
    }
```

```
    for (NSInteger i=0; i< m_iNumDynamicObjects; i++) {
        [m_pDynamicObjects[i] Render];
    }


}

#pragma mark ================================== OBJECTS MANIPULATION

- (void) AddUnitAt: (CGPoint)pos_ rot:(CGFloat)rot_   {
    if (m_iNumDynamicObjects < MAX_DYN_OBJS) {
        m_pDynamicObjects[m_iNumDynamicObjects] = [[UnitController alloc] initUnit:m_iNumDynamicObjects++
pos:pos_ rot:rot_ world:self];
    }
}

/* AddUnitAt: rot: withDNA:
 - used when units breed to create a unit with a specific DNA string
 */
- (void) AddUnitAt: (CGPoint)pos_ rot:(CGFloat)rot_ withDNA:(NSString*)dna_ {
    if (m_iNumDynamicObjects < MAX_DYN_OBJS) {
        m_pDynamicObjects[m_iNumDynamicObjects] = [[UnitController alloc] initUnit:m_iNumDynamicObjects++
pos:pos_ rot:rot_ world:self DNA:dna_];
    }

}

- (void) AddDynamicObjectAt: (CGPoint)pos_ rot:(CGFloat)rot_ type:(Byte)type_ {
    m_pDynamicObjects [m_iNumDynamicObjects] = [[DynObjController alloc]
initDynObj:m_iNumDynamicObjects++ type:type_ pos:pos_ rot:rot_ world:self];
}
- (void) AddStaticObjectAt: (CGPoint)pos_ rot:(CGFloat)rot_ type:(Byte)type_ {
    m_pStaticObjects[m_iNumStaticObjects] = [[StatObj alloc] initStatObj:m_iNumStaticObjects++ type:type_
pos:pos_ rot:rot_];
}

#pragma mark ================================== TOUCHES
- (void) HandleTouchesBeginAt:(CGPoint)loc_ {
    for (NSInteger i=0; i< m_iNumDynamicObjects; i++) {
        [m_pDynamicObjects[i] HandleTouchesBeginAt:loc_];
    }
}
- (void) HandleTouchesMovedAt:(CGPoint)loc_ {
    for (NSInteger i=0; i< m_iNumDynamicObjects; i++) {
        [m_pDynamicObjects[i] HandleTouchesMovedAt:loc_];
    }
}
- (void) HandleTouchesEndedAt:(CGPoint)loc_ {
    for (NSInteger i=0; i< m_iNumDynamicObjects; i++) {
        [m_pDynamicObjects[i] HandleTouchesEndedAt:(CGPoint)loc_];
    }
}

#pragma mark ========================== OTHER
/* GetDynObjWithId:  - returns pointer to a dynamic object of a given array id  */
```

```
- (DynObjController*) GetDynObjWithId:(NSInteger)id_ {
    return m_pDynamicObjects[id_];
}

/* GetDynObjAt: - returns array id of a dynamic object at a certain position
                - returns -1 if no object found  */

- (int) GetDynObjIdAt: (CGPoint)pos_ {
    int returnVal = -1;
    for (int i=0; i< m_iNumDynamicObjects; i++) {
        if ([m_pDynamicObjects[i] HitTest:pos_]) {
           returnVal = i;
        }

    }
    return returnVal;
}

- (void) UnselectAllOtherObjects:(NSUInteger)id_{
    for (int i=0;i<MAX_DYN_OBJS; i++) {
        if (i != id_) {
        [m_pDynamicObjects[i] Unselect];
        }

    }
}

#pragma mark ============================== PERSISTENCE
- (NSString *) dataFilePath {
    //--- look for the Documents directory within the application's sandbox
    NSArray  *paths  =  NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,  NSUserDomainMask,
YES);
    NSString *documentsDir = [paths objectAtIndex:0];
    //------- append the filename to get its full path:
    return [documentsDir stringByAppendingPathComponent:SETTINGS_FILE_NAME];
}

- (void) applicationWillTerminate: (NSNotification *) notification {
    NSMutableArray *array = [[NSMutableArray alloc] init];
    for (int i=0; i<m_iNumDynamicObjects;i++) {
        [array addObject:[NSNumber numberWithInteger:m_pDynamicObjects[i].m_iType]]; // [REF 1]
    }
    //---------------- writing atomically -> only rewrite the file if save complete
    [array writeToFile:[self dataFilePath] atomically:YES];
    [array release];
}
@end
```

Appendix M: UML of the Implemented Architecture