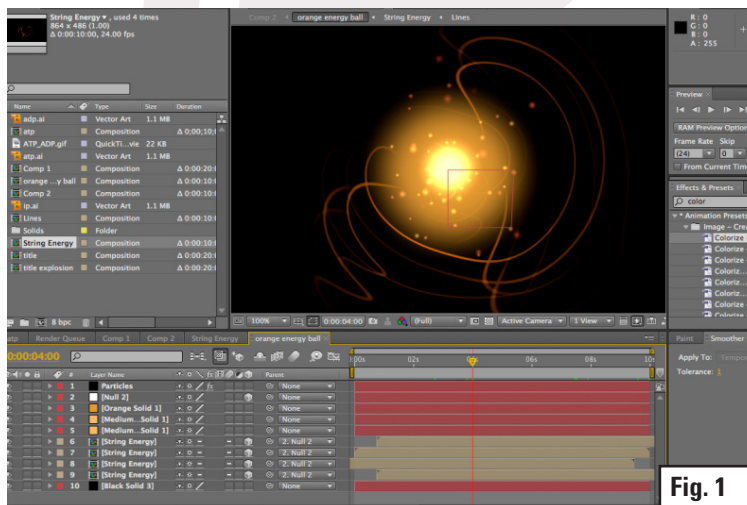




Meta!Blast: Behind-The-Game Technical Description

Jonathan Ackerman and Meta!Blast Developers

Meta!Blast (<http://metablast.org/>) is a product of the collaboration among a group of highly skilled and creative artists, game designers, computer programmers, and biologists. Together they insure it will be at once educational and engaging, and provide an accurate portrayal of biological phenomena in a gaming format that students will recognize and enjoy. For the group to create such a product, some of the biologists, writers, artists and programmers need to be conversant in each other's languages. Meta!Blast is being created from the ground up, so every aspect of the game is completely original and never seen before. The innovative computer code, artwork, and educational motive behind Meta!Blast can make the game and its development seem even more of a mystery to those already unfamiliar with the process of video game creation. In the hopes of demystifying this exceptional game, the coming sections provide an exploration of the integral parts of the Meta!Blast project, along with some of the goals, challenges, and obstacles which go along with its development.



1) The Kabala Game Engine

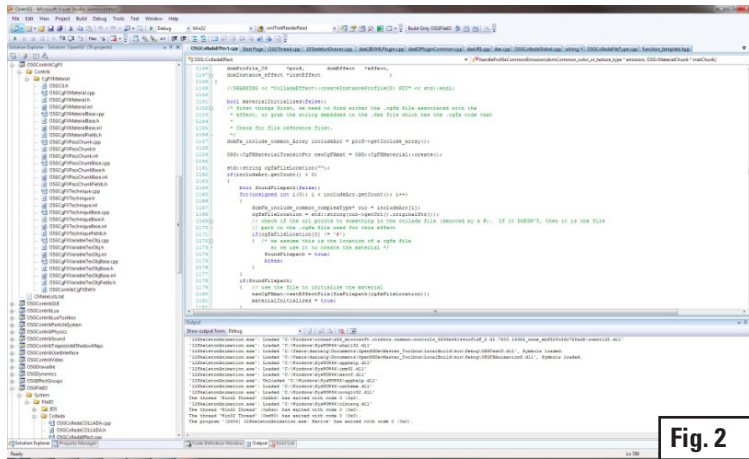
The first part of the process of game development for the computer programming team was to ensure the game engine is capable of supporting the capabilities of Meta!Blast just as its artists and developers envision it. Stated most simply, a game engine is a program to which art files and code from a scripting language can be 'passed', and which it compiles together to create the game the player sees. It is in many ways like the skeleton of a video game: just as a skeleton determines what sort of activities an animal might be able to do but cannot move on its own, the game engine provides the framework that will define a game's capabilities and limitations, but remains dormant without additional code from the scripting language to bring it to life. Just as

achimpanzee and a human being have similar skeletons but dramatically different abilities, so too can very different video games be built using the same game engine. (For more information on game engines in general, their history, and variants go http://en.wikipedia.org/wiki/Game_engine.)

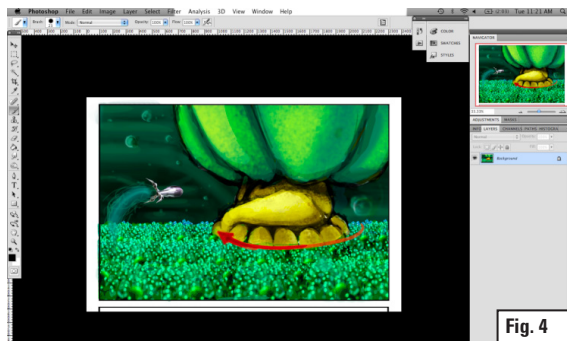
If two games are created using the same game engine, then what makes them differ from one another is the script code and art work that are 'passed' to the engine. You can think of the art and scrip code like the organs, muscles, and other tissues of a body: they are the moving pieces and unique visual features that define a game's features, like muscles, skin, and hair will make one person look and act differently from another. As one might imagine, the skeleton must be finished before these tissues and organs can be added, but once the engine is completed the programmers will move on to work in unison with the artists and game developers to swiftly form prototype after prototype of the game until a satisfactory version is produced. We are using two engines as the skeletons for the Meta!Blast project, Unity (<http://unity3d.com/>) for desktops and laptops, and the Kabala Engine (described below, Wurtele etal., WINVR2010) for multiple platforms, including a multi-display system. This document focuses on the Kabala Engine.

The Kabala Engine has been developed by David Kabala, head of the programming team, and 2-5 undergraduate and graduate students. New capabilities are added as needed. Aptly named the Kabala engine, it was developed for, among other things, its open source software and its ability to be easily integrated into many types of platforms, from a multi-display system, such as Iowa State's VRAC facility, to a simple MAC or PC. The engine is compiled in the C++ programming language. Most of its functions are in a set of core libraries. Such libraries include capacities for creation of a user interface, physics, sound, animation, art import, and a particle system (a particle system defines the movements of everything that moves independently from the players' actions). The term "library" might imply databases containing every variety of sound (for a sound library) or animation (for an animation library), such that a programmer could simply choose which 'setting' to use, however, this is not at all the case. The LUA language is used by the programers as the scripting language to augment the Kabala engine, and it is LUA that defines how these libraries are used. LUA is a non-compiling language, which means modifications to the LUA code made during prototyping can occur while the game is running, cutting down the time required to make changes to the game tremendously! Other languages were considered as possible scripting languages for Meta!Blast: Python, Ruby, and Java Script would all have been viable options. LUA, however, was constructed to be an embedded language, a language meant to be used inside a native language (in this case C++) application, and is smaller (requires less memory) and faster than most others. This, compounded with LUA's track record in the gaming industry (it was used as the scripting language in World of Warcraft), made it the best choice for Meta!Blast. More information about LUA can be found at <http://www.lua.org>.

Kabala and LUA interact to form the behaviors for Meta!Blast. Behaviors define the way the game will react when certain 'events' occur during game-play. Events are anything that trigger a response from the game; they can be as basic as a click of the mouse to as pivotal as defeating a boss. Behaviors, too, can take various forms, and can be quite blatant to nearly invisible. For instance, if a student playing Meta!Blast bumps her ship into, say, an organelle the game could respond with sound effects, vibrations, and explosions or a minute decrease in the player's health or both. In this case the initial event would be the bump into the organelle and the behavior the response.



One of the strengths of the Kabala engine is that it has a way to run behaviors generically, that is, regardless of the behavior Kabala will implement it using one streamlined protocol; a sort of hierarchical chain of command which will filter an event through the impenetrable jungle of computing code so that it reaches the proper behavior(s) destination. To begin with, in the Kabala Engine (that is, in the abstract C++ skeleton) behaviors are first attached to scene objects. These scene objects are essentially place holders for information which will move between the scenes of the game, lending continuity to game play. The player may eventually encounter scene objects in a recognizable form, such as energy level or a newly acquired ability, but keep in mind that at this point scene objects are still abstract and require LUA code before they can acquire specific identities and functions. Containing the scene objects are the game's "scenes", also still in their abstract form, which with the addition of the LUA scripts may become a room, the ship's deck, or the plant cell itself. It may be easier now to think of the game, scenes, scene objects, behaviors, and effects as the box and ball model in Figure 1. Within this C++ framework LUA scripts are called to do all of the decision-making for the game: deciding whether or not to run this or that behavior. Figure 2 gives a naive interpretation of how LUA and the Kabala Engine interact to manage Meta!Blast's behaviors.



2) Art

The task of inventing ways to make Meta!Blast visually exciting belongs for the most part to the Meta!Blast art team. The art team works closely with the biologists in the Meta!Blast project to ensure their ideas remain biologically sound. This means that not only do the cell structures and reactions have to look as "biologically feasible" as possible (some of the protein structures incorporated in Meta!Blast are not even fully understood by scientists today, while envisioning

light-reactions at the atomic level is essentially impossible!) but also the ship, gadgets and tools, and of course the nemeses all need to be as biologically-based as possible. So instead of guns, shields, and alien swarms, Meta!Blast players will have electromagnetic radiation emitters, chemically-powered engines, chemical reducing tools, and enemies that could be found in a futuristic medical journal. It is the artist's job to make these components just as impressive and fun as any modern video game gizmo. While most studio art departments for big name games such as Halo or Call of Duty have a bevy of artists, modelers, art leads and art directors to manage the many artistic aspects of a game, often with entire teams devoted solely to the characters, environments, vehicles, or effects, the Meta!Blast team has 2-4 student-artists at a time who are also taking classes and still learning digital modeling and animation, adding to their challenge. Thus intense collaboration with the other artists, scientists, and game developers is vital to their work.

The first stage for any artistic concept is most often a 2D drawing. or drawings in which the artist explores a character or design with many variations on a particular theme. The artist then meets with the game developers and biologists to discuss their designs, rejecting those that may not work and getting input on how to extrapolate on those that will. This process of creation, discussion and revision sometimes takes quite a while before an acceptable end product is achieved, but in addition to a final design, this process usually leaves the artist with an exceptional collection of concept artwork that she/he can add to his/her portfolio.

The final 2D designs are then extrapolated into 3D using the animating program Maya followed by another rigorous process of refinement. Maya is a professional drawing program that has been used extensively in the gaming industry for games such as Kill Zone 2, the Uncharted series, and Halo (www.maya.com for more info). The 3D structures begin as geometric objects made of 'polygons', simple triangular shapes that determine the intricacy of the structure they define. After the basic geometrical structure is created another set of refinements is implemented. Textures, the wide variety of calculations that are used to create a visual representation of the effects of light, shadow, and color on the surface of the 3D object, are then applied to the 3D models. The final step of the 3D art creation process is rendering, in which all of the elements of the image are processed and brought together in a high-resolution format. Figures 3-5 illustrate the process of Meta!Blast artists using Maya.

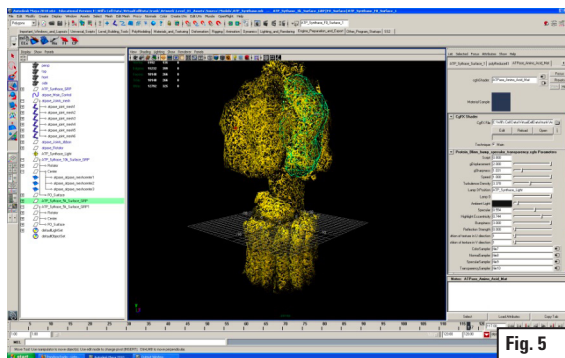


Fig. 5

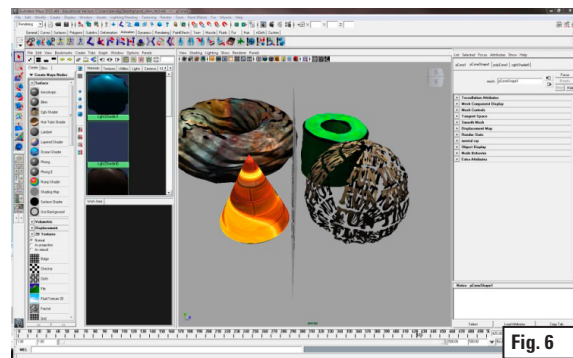


Fig. 6

The artists face a few technical restrictions to their designs: the scenes (in this case a "scene" is the vista that a player would see at a timepoint in the game, and not a "scene" as defined in the Engine section) need to be kept below 100,000 polygons. Also, they needed to be wary of the

number of “textures” used, as well as the resolution of those textures. Keeping to these requirements insures that the game’s art would not eat up too much memory - slowing the game play intolerably.

A finished piece of Maya art is exported into a Collada file (<http://www.collada.org/>) using Maya’s exporter. This process allows Kabala’s importer to recognize and include it in the game. Importing the collada file occurs according to the scene graph of the image, which is a set of vertices and edges that branch out like a tree from a central vertex. Each vertex represents a ‘level’ or ‘layer’ of the scene (see figure). The central vertex, or root, is usually the light source, which is, in a sense, the foundation of every scene. Each subsequent vertex could be anything from a type of movement, to a geometric shape, to a texture. The Kabala importer ‘walks’ down the scene graph, beginning from an initial vertex and descending to a connected vertex below it, importing the ‘layer’ each vertex represents as it goes. If a vertex has multiple vertices below it, the importer will choose one to move to, follow it down until it cannot walk any further, then go back to that initial vertex and choose a different vertex to move to. This always-descending method of importing is called ‘depth-first’ importing.

3) Game Developers

The game developers’ challenges and goals are diverse; because of the comparatively small of the Meta!Blast team, they are often working side by side with the artists, and programmers on the nitty-gritty aspects of the game while at the same time developing the big picture aspects a game of this complexity entails. The most difficult problem facing the development team, perhaps the central question of the entire project, was how to make a truly great educational video game. Each division of the team has their own ideas and contributions to make towards a solution to this problem.

3.1 The Teachers

For the high school and middle school teachers involved in this project, this is a challenge of personal importance as they are the ones putting the finished Meta!Blast game to the test in their own classrooms. For them, incorporating teacher involvement into the game was a high priority. Students left to their own designs with the game may or may not take from it the educational information it contains, that is they may play it as if it were any other video game; a teacher’s presence and guidance would be needed to insure that the students absorb the games deeper, biological and educational content. To this end, the teachers brainstorm ways to make Meta!Blast a teaching tool, and also ways to keep teachers involved and able to use Meta!Blast efficiently and diversely in the classroom. In their search for solutions, the teachers consider the technical limitations of the game engine, and check with the programmers to make sure the complexity of their ideas does not transcend its computing capacity. For instance, one idea to network all the computers in the classroom so that the teacher would have some degree of control over all the student’s game experience from a single ‘master computer’ might prove too computationally expensive for the Kabala engine to feasibly run.

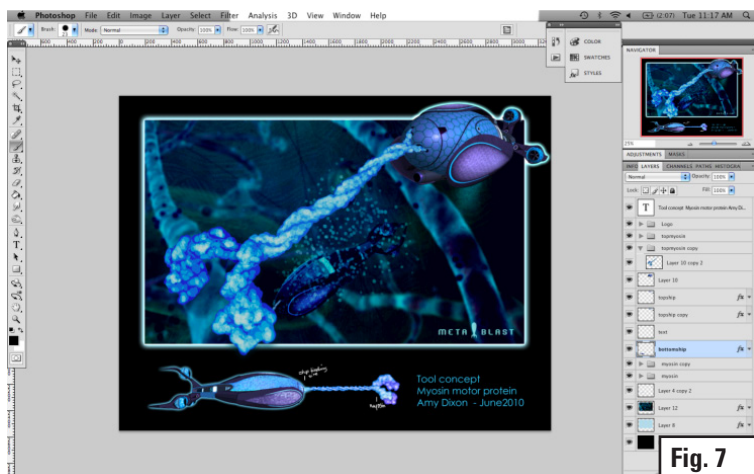
3.2 The Writer

The writers work with the rest of the team to come up with a biologically-based piece of science fiction (yes, biologically-based science fiction is an oxymoron) that is exciting, involving and educational for the players. One major challenge is to use technical terms but avoid boring the player, and to keep the biology coming. Another major challenge is for the writers to learn the

complexities of biology.

3.3 The Biologists

Students are taught highly complex concepts that cannot be directly visualized, and Meta!Blast was conceived as an approach to address education on metabolic and cell biology. Biologists act as both contributors and editors for the Meta!Blast project. While they, along with pretty much everyone else on the team, are encouraged to bring their ideas for game play to the table for discussion, their main purpose is to help Meta!Blast retain accuracy to real biological processes. They use, for instance, the program PyMol (see <http://www.pymol.org/> for images, history, and uses), a scientific program that helps to predict the shape and function of a molecule, to transfer to the artists a sound picture of what the parts of the cell should look like. They also attempt to avoid the standard game interaction tools, such as a “tractor beam”, “laser gun”, “missile”, etc..., and instead find biological analogues which both help to teach and better fit the gameplay. For example, an alternative to a “tractor beam” being used in the Meta!Blast game is a genetically engineered myosin motor protein that can be controlled by the player to grab and move parts of the cell (see Figure 7).



DEFINITIONS

Game Engine: The program through which the player experiences the game.

Behaviors: The pre-assigned and networked actions a game will take when certain conditions are met during game play.

Rendering: is the final process of creating the actual 2D image or animation from the prepared scene. rendering entails the application of all textures to all dimensions of the 3D models, in full resolution Fig. 7

Real-time Rendering: The scene content is calculated and displayed at rates of approximately 20 to 120 frames per second. In real-time rendering, the goal is to show as much information as possible as the eye can process in a 30th of a second. The more polygons in a scene, the longer it takes to render. For each game engine, the time it takes to render a given scene will differ.

Scene Graph: A set of vertices and edges that branch out from a central vertex. Each vertex represents another 'level' or 'layer' of the scene.