Arcane Arena Document
By Brandon Keyes and Jeffery Nehring

R2.1) Introduction: provide a brief description of the game, genre, objectives, style of play, background information. Here is a great place to pitch why your think your game is great. Make us want to play it! Make use of screen shots to help explain and illustrate your game. Briefly introduce the game objectives and flow. If your game resembles, or is based on, an existing game then mention that.

Arcane Arena is an arena-based 2D survival shooter designed to slowly pressure the player into making mistakes. Mistakes get you killed.

Illustration 1: The river separates the player from the enemies.

The player must survive the rapid onslaught of enemies through the use of tactical play. While the early levels don't require much in terms of the skill of the player, later levels increase the spawn rate of enemies until it is too much to handle. In later levels, projectiles might be fired by multiple enemies simultaneously in an unpredictable fashion at the player. Evasive tactics must be adapted by the player as difficulty ramps up, all while battling enemy mages that have an unpredictable evasive pattern and a mana system that rewards accuracy from the player.

The player accrues score for hitting enemies. Hitting enemies with a basic attack gives less score than with a special attack. Survive as many levels as possible and attain higher and higher scores as your skill increases. A mana boost is given to the player every time they land a basic attack, but special attacks take a ton of mana to use, so accuracy is very important with these attacks.

Illustration 2: Later levels bring more enemies.

The game is based on Magicka, a game similar in style and play, only tweaked for arena-based gameplay.

R2.2) Installation: describe the software/hardware environment needed to install and run your game. Provide specific installation instructions. We will want to do this. The marker is going to be following your instructions and installing and testing your game. If your game cannot be installed per your instructions further evaluation will not continue. Make sure to point out any special concerns or requirements regarding installation. You can assume the marker will start with a default install of Visual Studio 2012 and the Windows SDK. You must describe where to find and install any additional components needed.

Hardware needed: Windows 7/8/8.1 OS, x64 (64 bit) architecture.
Software needed: all additional software is included already.

There are two ways to run the game. Either launch the 2501Game.exe in
C:\...\2501Game\2501Game

Or through the debugger. There are two steps to follow before the debugger will work.
First, click 2501Game in the Solutions Explorer. Then click Project → Properties → Configuration Properties → C/C++ and select Additional Include Directories. Click the drop-down arrow and select Edit. Here you're going to select the folder icon, a blank space should have appeared in the pane. Select it, a button with three dots should have appeared. Click it, select the “Inc” folder in the 2501Game directory. That's the first addition include directory. Do the same process for the “include” folder in the SDL2 folder and the SDL2_mixer folder in 2501Game.

Now we need to add the lib folders from SDL2 and SDL2_mixer. Navigate to Linker under Configuration Properties, then to General. Select Additional Library Directories. Add a new directory and select 2501Game/2501Game/SDL2/lib/x64 as the first include, and 2501Game/2501Game/SDL2_mixer/lib/x64 as the second.

Debugging will now work.

R2.3) Start Up: Describe how to launch your game and get ready for play.

Again, two ways to launch. Either through the debugger in Visual Studio or by launching the executable (explained above). Upon launching, you'll be brought to a simple menu screen. You have three options, play the game, go to the options, or exit. Options allows you to mute all music and sound effects.

R2.4) Rules and Interactions: Explain the rules of the game and user interactions with the game. Explain any options that the user has regarding user input devices and gestures. Explain any tutorial material or levels included with your game. Explain the game objectives from the player's perspective and how they progress through the game. For user interaction explain, and illustrate, carefully exactly how the controls work. Include diagrams where appropriate.

There is only one rule, and it is “survive”. The user has two ways to do this, by avoiding all incoming projectiles and by killing enemies.

The player can use WASD to move. 1, 2, 3, 4, 5 to select a special ability. Left click to fire the basic ability and right click to fire the special ability.

Press P to pause the game and view the pause menu, where further options are presented (mute music and sound effects or quit).

R2.5) Levels and Episodes: Explain what levels or episodes are build into the game and how one progresses from one to the other.

Levels increase at a fixed rate, while enemy spawning increases faster and faster as levels get higher.

R2.6) Play: Explain in detail how to play your game. Any strategy tips. Anything the user needs to know that has not been covered by the previous requirements.

Move around a lot. If you stay near the river you need to be aware that you don't have a lot of time to move out of the way of incoming projectiles, but landing attacks will be easier. Never fire at the enemy, fire around them. Basic attacks don't cost anything and landing them gets you bonus mana, fire those like crazy. Save your special attacks for when an enemy comes near the river.

R2.7) Scoring: Explain how scoring works and any options the player has for saving state at the end of the game.

There is Score and Level. The highest Score and Level are automatically saved, so the user need not worry about that. Score is the bonuses gotten from doing damage and killing. You get 2 points for hitting an enemy with the basic ability, 10 points for hitting and enemy with the special ability, and 25 points for killing an enemy.

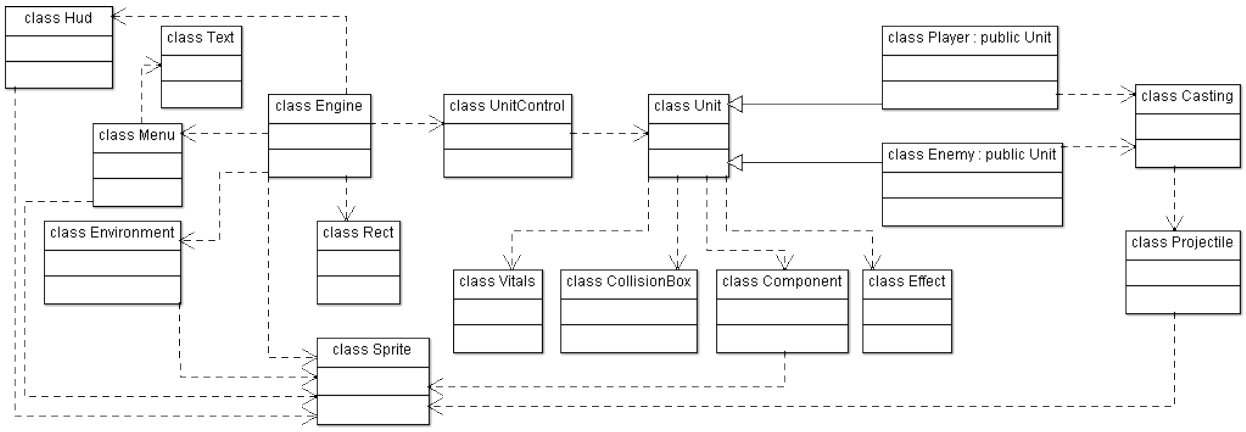
The Level dictates difficulty, the higher the Level the higher the enemy spawn rate.

The Score and Level are indicated during gameplay in the top left of the screen, Score is beside the S, and Level is beside the L. The highscore and highest level is indicated in the bottom left of the screen as HS and HL.

R3.1) Describe the overall software architecture of the game.

The game is broken down into a few parts. Enemy Spawning, Entity Movement, Collisions, Music and Sound, and Drawing. They are all updated (when running) in the same order listed.
Enemy Spawning is handled by a function called HandleEnemySpawning which spawns enemies based on a couple factors, but mainly the current level. A higher level gives a higher spawn rate.
Entity Movement is handled by the HandleAllMovement function, which moves the enemies and the player, and handles all projectile movement.
Collisions are handled in the HandleAllCollisions function. Player projectiles are checked against enemy collision boxes, and enemy projectiles are checked against the player's hit box. Consequences of collisions, such as point bonuses or damages are handled here as well.
Music and Sound is handled in the HandleMusic function. Music is stopped and started here based on information retrieved from the Menu class.
Finally, Drawing is handled by the Render class, this draws everything needed to be seen by the player. Enemies are spawned in this function as well, but mostly

R3.2) Provide a UML style class diagram to show how your classes are organized. (It does not have to follow exact UML syntax -basically boxes for classes and arrows show relationships. Label the relationships so we can see what the mean.) Provide a short description of each class -its main roles and responsibilities within your code. Do the major classes if you have a lot.



R3.3) Data Structures and Performance: Describe the important data structures used to represent aspects of your game. Describe any performance issues you faced and how they were addressed in your design.

We used vectors from the standard library for all data storage. Collision checking required fast access to data to check members quickly and efficiently. Removing data efficiently was of lesser concern (so array deques were out of the question). When checking vectors that contained projectiles from the player and enemy they were passed by pointer so as not to spend too much time copying data, this meant that data was read efficiently. As for performance issues, we initially used an array deque assuming adding and removing would be the most costly for performance, but as the game got larger this was proven incorrect. Also, vectors containing projectiles were initially passed by value, which caused collision detection to be costly. Once changed to vectors passed by pointers, collision detection ran more smoothly on lesser performing computers.

We used a vector of vectors to contain the different sound effect files. When casting, each different ability has three separate sounds that are randomly used to signify the casting sound. This is done by having a vector contain more vectors that each represent and ability.

R3.4) External Libraries And Artwork: Describe what additional libraries or toolkits beyond just Visual Studio 2012 and the Windows SDK use used. (e.g. if you used DirectXTD or the DirectXSDK associated with Visual Studio 2010). Explain what the library provided that you needed in your game.

We used two additional libraries. These were SDL2 and SDL2 Mixer, solely for the purpose of integrating music and sound effects in the game.

R3.5) Tools: Describe any tools you used beyond just Visual Studio. Provide information on what the tools do and how they were helpful to you. Also describe any tools your built, or think you should have built, to aid the construction of the game. Describe what tools you used for the artwork of your game.

Linux MultiMedia Studio and Hydrogen were used to create the music tracks for the game (the sound effects aren't original). Linux MultiMedia Studio allows you to create your own music, where you can use orchestral plugins, acoustic plugins, piano plugins and more. Hydrogen allows you to create a drum beat using their drum machine and export it to LMMS.

Photoshop was used to make all of the art assets.

R3.6) Difficult Parts: Describe which aspects of your game development you found the most difficult, or challenging, to design or implement. Describe any unforeseen obstacles and how you overcame them. (This will be helpful for us to plan tutorials in the future.)

There were a couple challenges along the way. One of them being projectiles. Initially projectiles were added to an array deque so that they could be easily deleted, but we had so many projectiles going at once that accessing their properties became a performance issue, so we switched to a vector instead.

There are a lot of moving entities on screen in later levels, and on weaker machines it can be difficult to run it all. Collisions are the primary resource hog, so they are performed at a lesser rate than drawing. We focused on collision checking efficiency. Collisions are checked between the player's projectiles

and the enemies, and then the enemy projectiles are checked against the player. Projectiles are only checked if they are across the river. Then collisions between projectile and entity are checked not by checking if the projectile is inside the entity, but outside of it. This way, the max checks needed to be sure a collision occurred is 4, but the least required to determine a collision didn't occur is 1 (for example, the projectile is too far to the right of the collision box, no collision can occur, stop checking).

R3.7) Special Features: Describe any aspects of your design that you feel makes your game or approach unique. Anything that you feel is special that a programmer wishing to continue work on your game should know.

We've made an effort to keep everything modular. We've succeeded for the most part. Maintaining this is important for the overall code to function properly and the game to perform it's best.

R3.8) Incomplete: Explain which parts of the game were not completed to your satisfaction and roughly what is required to complete those parts.

The menu and score/level parts of the HUD are rather lackluster, but we were pressed for time. They are functional and work great, but they aren't perfect. Of course, the actual gameplay is most important but it'd be nice to have a consistent product. To fix this we'd just need to create more graphics for the score/level portions of the gameplay screen and menu.

A feature we would have liked to implement if we had more time was dismemberment. This would have been done by setting each enemy's individual parts (hand, head, body, etc) as an individual collision box and then handling collisions between each one separately. If one part took too much damage, the enemy would lose that part, and losing a part would have a consequence, such as slower movement or casting.
We didn't have time to add projectile effects based on the type of projectile used. We would have liked to have enemies and the player be effected by the different types of projectiles in different ways.