# Run! Design Document

an Action Escape Game from Dragon Arts

Dragon Arts – Game 350 B – Spring 2009

2/20/2009

## Table of Contents

## High Concept

A 3rd person, fast-paced, action escape game with emphasis on speed, style and continuous momentum.

## Target Demographic

Gamers of both genders familiar with 3D actions games, ranging in age from 10 to 50+. ESRB Rating: **E**.

## Target Platform

The target platform is Windows XP and Windows Vista PC computers.

## Purpose

To allow the player to experience Parkour in its essence, move swiftly, elegantly and efficiently, overcome obstacles not found in everyday life and enjoy the humorous misadventures of youth.

## Minimum System Requirements

[TBD]

## Product Comparison

### Assassin's Creed

The fast-paced, frenetic speed and thrill of the chase that Altair experiences in *Assassin's Creed* is a strong influence on *Run!*. Differing from *Assassin's Creed*, *Run!* will contain no combat, adult themes, or open world between missions. It will also focus more on running over rooftops rather than stealthily blending with the crowd.

### Tomb Raider: Legend and Tomb Raider: Anniversary

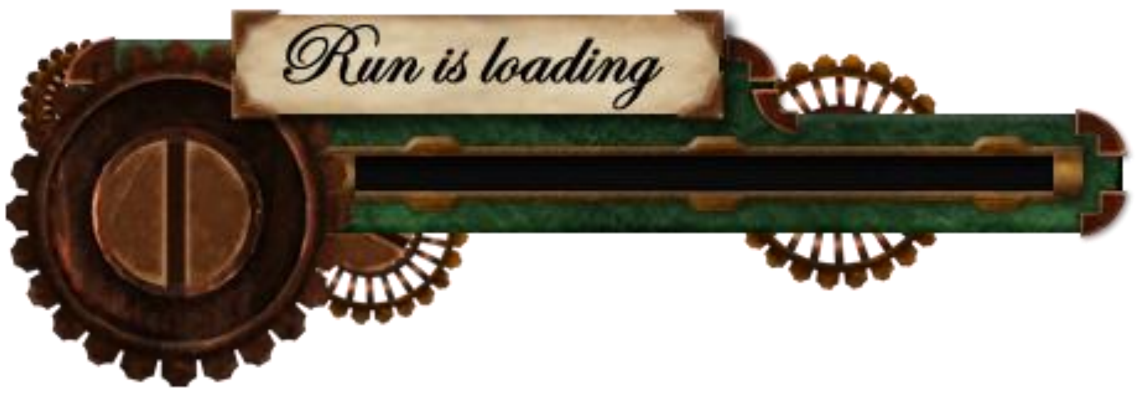
*Tomb Raider* influences *Run!* on the climbing interactions side of things. The use of very specific move-sets and a well defined set of terrain features that may be interacted with will play an important role in *Run!*. These game design features are heavily influenced by both of the latest *Tomb Raider* games.

## Example of Play

The player has already installed the game to their computer.

The player double-clicks on the application shortcut on their desktop.

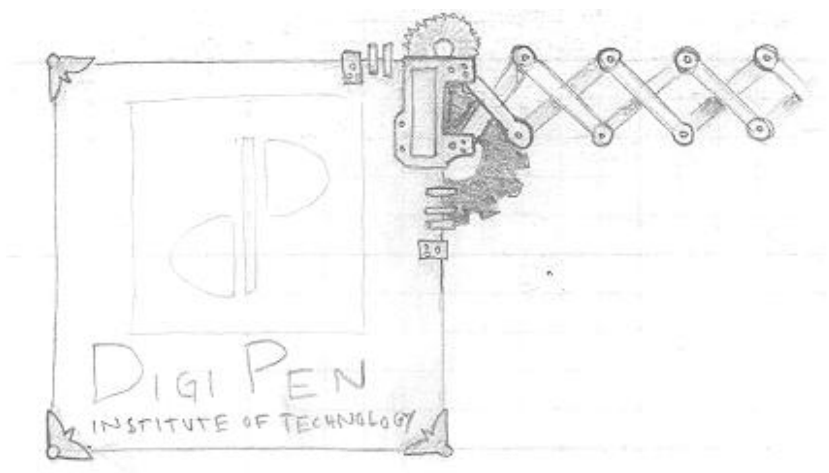
The following appears in the center of the player's screen:



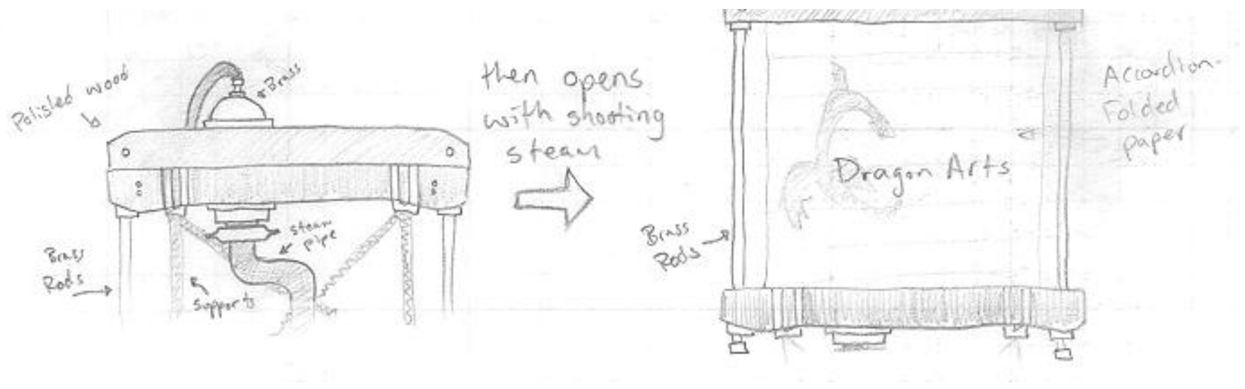
The bar has a piston in it that fills up as the game loads. The gears turn. As the game loading nears completion, the gears turn faster and steam jets from behind the apparatus. As it completes more steam jets from behind and fills the screen to white.

If the game is set to start in full-screen mode, the game starts. Otherwise, if it is set to start in windowed-mode, the white screen shrinks over one second to the size of the window specified by the user. The game proceeds as follows:

The following logo springs in from the side:

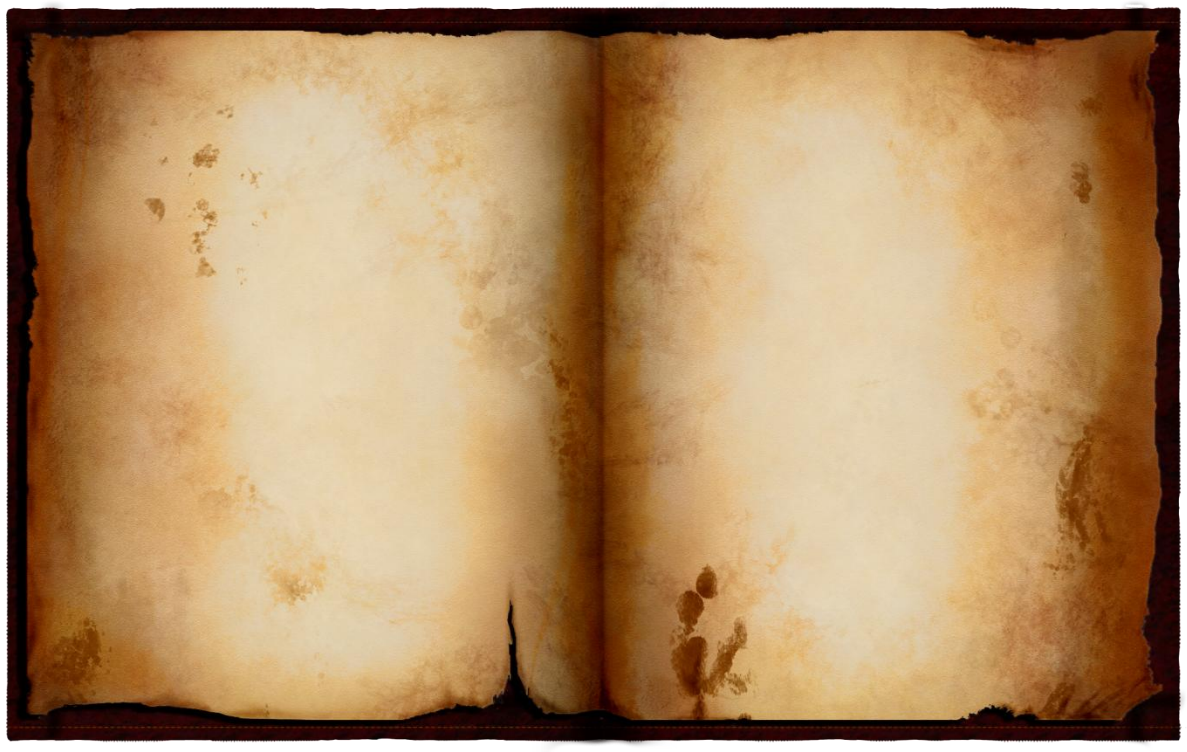


The logo hangs for a few seconds, or until the player clicks on it, and then it breaks, gears and sign falling off. The spring slides back off to the right. Then from the bottom the following sequence plays:



After a few seconds, or when the player click on it, the sign slides off the bottom of the page.

The scene fades in from white and opens with a view from a rooftop overlooking the Victorian city. Smoke billows from many chimneys. Many houses have metal plating instead of brick or wood for side. Victorian architectural features are prevalent everywhere, as well as bolts, gears and other mechanical or clockwork pieces. Pipes of all sizes and shapes arc, twist and extend from building to building. Our hero stands, arms akimbo and a confident smirk on his face. The menu appears on the left, like an open, hand-written book.



There is careful, cursive script marking the main menu options. Alongside the calligraphic script are hastily written remarks, comments and smudges, obviously made by the Hero's hand. The player uses the mouse to click on an item, or the Xbox 360 controller left-control-stick or d-pad to maneuver the cursor between choices and press the A button to select. The player may choose to start a new game, continue their current game, change options, change the current profile or exit the game.

If the player chooses to change their options, they are presented with options to change graphics settings, audio settings, and gameplay settings.

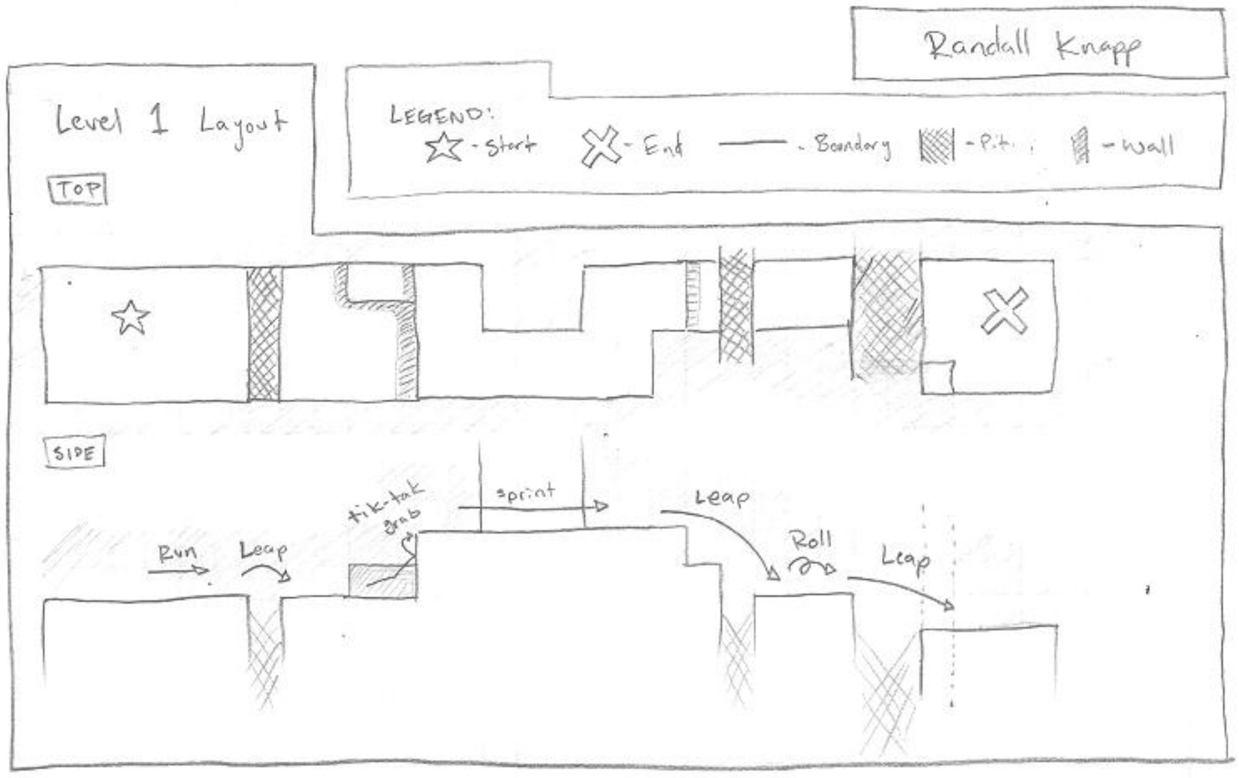
If they player chooses to change profile, they are presented with a list of available profiles on their computer, and they may choose to select, create or delete profiles.

If they player chooses a new game, and they have not created a profile yet, they are prompted to do so. Once they have created a profile the menu fades to black and the game starts.

If they choose to continue their game (and the option would only be available if they have a currently started game) the menu fades to black and the game starts immediately.

The main character turns and starts running, and then freezes. The camera pans down the rooftops towards the goal, circles around it once and pans back to behind the main character. The action resumes and the player begins playing.

Here is the level layout for the first level:



The player pushes forward on the control stick (or W key) and the character moves forward and begins running. They approach a short gap and press the Jump button (A button or Space Bar). The character leaps forward without losing momentum. Just as he is about to clear the pit, the player presses the

Action button (Right Trigger or Left Mouse Button) and the character dives forward into a roll and comes out running, giving the player a short speed boost. This is shown by air stream trails flying beside the character.

The player now reaches a ledge that is taller than the character, with a low wall to the left. The player presses the action button as they approach the lower wall, and the character jumps off the left wall and plants a foot on the tall wall, and when he reaches the top, the player presses the action and roll buttons (Left Trigger or Right Mouse Button) simultaneously. The character grabs the ledge, hauls himself up in a fluid motion and tucks into a roll. He comes out of the roll and gets another speed boost.

The player now sprints across the rooftop, swerving a little out of the way of a big chimney. The character comes to the ledge, and is moving really fast. The player presses the jump button. The character leaps over the gap. As he is doing so, the camera swivels around to the side and the game slows down. They player is awarded a slow motion view of the character as he lands on the other side and continues on. He leaps the last gap towards the goal and reaches it.

The screen blurs out and freezes. A papery looking UI panel comes into view and shows the player the time they took to complete the level, the style points they were awarded and an encouraging statement leading them to the next level.

## Story

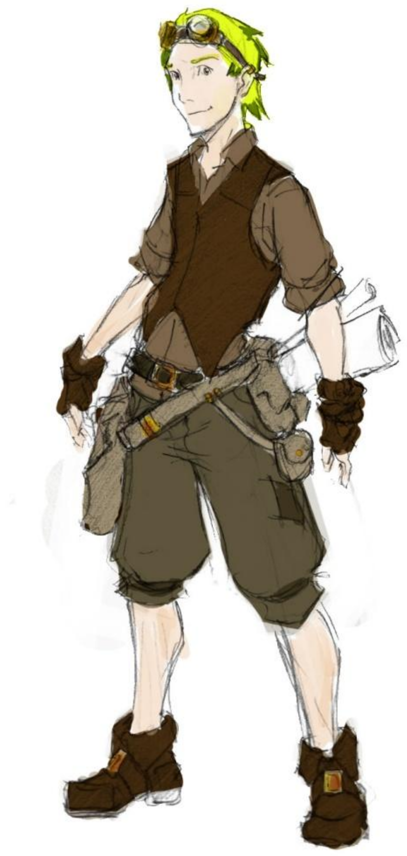
### Setting

*Run* is set in what we call a Steampunk Victorian City. As described in the example of play, the city looks like London or another large city in England or Europe during the Victorian era. Peaked roofs are common, but many flat industrial type roofs are present as well. The city seems in total juxtaposition of old and new, of tradition and technology. Buildings might be beautiful, graceful and artistic and be built right next to a large, metal-plated factory, belching smoke into the sky from many chimneys. Bellows and boilers are common rooftop or alleyway sights, and pipes of all sizes, color and material run, branch and extend all over every building, across every alley and above every street. The player is limited to the rooftops only. They may not descend to the streets for any reason. This is so we do not have to model people or simulate a busy city. We can do this however, with sound.

### Characters

#### Aldus "Trace" Traceur (Male Protagonist)

*Troublemaker and Ladies' Man*



The main character is a young man, around 22 years old. He is smart, clever and more than a little snarky. He appears as a medium high, narrow, lithe built pale-skinned man with blond hair. He is wearing tan trousers gathered just below the knee, a long-sleeved button-up shirt rolled up to the elbows with a brown vest over the shirt.  He is also wearing a pair of leather, brass and glass goggles pushed up onto his forehead and fingerless climbing gloves.  Round his waist are several belts, bags and pouches.  He is a charming, handsome young man with a smooth voice and appealing smile.  He is overly outgoing and has a slightly rash and overzealous streak.

### Lenora (Mayor’s Daughter)

*Beautiful, elusive and quintessential*

This lovely young lady is the subject of our Trace’s attention and affection.

[More on her is TBD.]

## Plot

The plot of the story revolves around the Trace’s quest to woo the mayor’s daughter, Lenora.  Lenora secrets enjoys the affections and lengths that Trace goes to for her, but she is constantly asking him to improve, do one more thing, or bring more proof of his affection. Trace will be running all over the city in attempt to impress this lovely vixen.

# Game Mechanics

## General

### Chapters

Chapters are our name for levels or missions. During each chapter, the player will be shown the goal and the general path of how to reach it. The player jumps, run and flips their way to the end. Once the player reaches the ending region, the chapter ends and they are shown their time, style score and previous bests.

### Time

Each chapter will give the player an allotted amount of time to reach the goal. They must reach their goal inside the time limit or they will fail and get “caught” and must restart the chapter.

### Searchlights/Steam-copters

When you start the Chapter you be informed that the authorities are looking for you. Their search is the based on a spotlight from some kind of steam flying machine, an Airship or steam powered helicopter. If the player stands still for a couple of seconds they will notice the lit area of a spotlight a couple of buildings ahead of them. The light won't ever just appear on the player, but it will always move slowly toward them as if the authorities are searching. The player can run faster than the authorities can search so if they keep up their momentum they will not be discovered. However, if the player doesn’t move the light will definitely find them. If they get spotted the player still has some time to run away and lose the light, but if they get really tripped up and don't move fast enough out of the light the player will lose the chapter. The lights are mounted on two kinds of steam-copters. One is a small light mounted on a small, comical looking clockwork flying gizmo. These are out often in the later missions and will serve as a motivation to keep running. If one of the little steam-copters “catches” Trace, a large, scary steam-copter will begin to chase Trace.

### Win/Lose Conditions

If the player reaches the goal point or area inside the allotted time, they win. If they player falls to the streets, is caught by the searchlights or steam-copters or does not get to the end when the time runs out, they lose, and must restart the level.

### Movement

Movement can be divided into two categories: running and climbing.

- **Running**
    - The character runs quite fast, he is essentially sprinting all the time
    - While running the character may vault, run on walls or roll. These are still considered part of running. See the Move Set below.
    - The player may change the direction of the run by rotating the camera or turning the character. See controls below.
- **Climbing**

- Climbing is whenever the player is hanging from a ledge.
- See Move Set below.

The character often changes between running and climbing. The transitions are smooth and do not break momentum.

**Falling:** When the character falls into the streets, they fail the mission and must restart. If they fall a long distance and do not recover properly (see recovery), they become injured for a short amount of time. They may not move while injured, and must wait until recovered to continue.

**Recovery:** If the character falls more than 1 story he must roll at the end to maintain momentum and avoid injury. The player must press the roll button as the player is approaching or hitting the ground to execute the roll and recover. If the player does not, the character injures themselves (temporarily) and must wait an agonizing second of inaction before continuing.

### *Move Set*

This is a complete set of moves. [Verbal and Visual descriptions TBD]

- Standing
- Walking
- Running
- Standing Jump
- Leap
- Wall Run (Along)
- Wall Run (Up)
- Wall Kick
- Jump Fall
- Jump Hang
- Mantle
- Medium Vault Up
- High Vault Up
- Medium Roll Up
- High Roll Up
- Cat Walk
- Cat Walk -> Jump
- Hurdle
- Vault
- Roll Vault
- Leapfrog Vault
- Medium Kick-Off
- Hang
- Full Hang

- Vault -> Hang (or Full Hang)
- Hang Transfer
- Hang Jump
- Diving Roll
- Recovery Roll
- Landing
- Falling
- Pole Hang (Parallel)
- Pole Hang (Perpendicular)
- Pole Grab (Perp.)
- Pole Swing (Perp.)
- Pole Jump (Perp.)
- Pole Hand-over-hand (Par.)
- Pole Step (Top)
- Pole Balance Walk (Top)
- Full Hang -> Hang

### *Saving/Loading*

The player will not be able to save and load during a chapter. However, when they reach certain parts of the level, they will save a checkpoint, and should they fail later in the level, they will return to the last activated checkpoint. They may alternately restart from the last checkpoint in the menu.

## Camera

The camera is a 3rd person spherical camera. The camera stays focused on the character (or tries to) as the player moves around. The player may move the camera using the mouse or the Right Stick. The camera will also move on its own. It will attempt to follow the player, constantly swinging behind where he is facing. Also, if the player moves very fast, the camera will get lower to the ground behind the player and change its field of view so that an illusion of extra speed is created. During incredible stunts or endings of chapters the camera will also swing into a "cinematic" view of the character and slow motion will go into effect so the player can enjoy their stunt in style.

## Controls

Controls will be customizable by the player. The following are defaults.

### *Keyboard & Mouse*

- WASD – Movement Forward/Back, strafe Left/Right
- Space – Jump
- Mouse – Look around, move camera
- Left Mouse Button – Action
- Right Mouse Button – Roll
- Escape – Pause
- Backspace – Back

### *Xbox 360 Controller*

- Left Stick - Movement Forward/Back, strafe Left/Right
- A – Jump
- Right Stick – Look around, move camera
- Right Trigger – Action
- Left Trigger – Roll
- Start – Pause
- Back or B – Back

## Menus

### *Main Menu*

This menu appears at the start of the game and any time the player returns to the main menu. It appears as an open book or journal with script labels. Messy handwriting, doodles and smudges made by the Main Character little its page. The options are as follows:

- Continue Game
- Start New Game
- Options
- Select Profile
- Exit

### *Options Menu*

The options menu is reached through either the main menu or the in game pause menu. The options are as follows:

- Graphics Settings – [TBD]
- Audio Settings – [TBD]
- Gameplay Settings – [TBD]

### *In-Game Pause Menu*

This menu is viewable while playing a chapter. The options are as follows:

- Restart Chapter
- Restart from last Checkpoint
- Options – Open the options menu
- Quit to Main Menu
- Exit Program
- Resume Game

### *Save / Load*

While playing, the player will frequently cross checkpoints, and should they fail later in the level will restart at the last checkpoint they crossed.  From the menu, they may choose to restart from the last checkpoint.

## Goals/Rewards

### *Best Time*

After each chapter is complete, the game stores the best time in the player's profile.  When the player completes a level they will be notified if they beat the latest best time [TBD: and see a current worldwide best time (via the website)].

### *Style Points*

An optional goal for the player will be Style Points.  The player *could* get from building to building just by using the analog stick and the jump button, but this may require them to avoid obstacles, cause them to slow down, or make them take longer routes. The game offers the ability to use stunts and hit those obstacles head on.  For instance, if they player is running towards a waist high obstacle, they can always just jump over it.  Boring! The game allows the way more stylish rolling vault!  Or, if the player is running for a wall too high for them to jump up, they would normally run right into it.  Fortunately, they can take the more stylish approach and run up the wall and back-flip to the next ledge. Boo yeah! The more stylish the stunt, the more style points the player receives upon completion.  Other stunt options include running along walls, doing flips or spins during long jumps or navigating a particularly difficult and intriguing piece of architecture.  Whenever the player chooses to do a stunt they are rewarded with Style Points that are accumulated into the player's profile [and posted to the website].  As a further bonus, the player will be rewarded with a speed boost, and, because his style is so cool, from off in the distance a certain young lady's voice exclaiming "Ooh!" or "Ah!".

## Game Environment

[TBD This is where the actual layout of each level would go and how they fit together to make the big city.]

## Game Objects

[TBD This is a list of all types of game objects that will be loaded into the world]

## Level Editor

The level editor seeks to minimize the need for complex structures to be built and edited within a modeling package.  It will take a 'Lego' style approach to constructing buildings in the city.  The editor will operate much like the building editor in *Spore*.  Starting with a simple building hull, windows, doors, and other embellishments will be added to the surface of the hull to create unique buildings for the level.

- Lego style approach to level editing:
    - Buildings begin as a simple hull
    - Hulls are arranged on the ground plane to begin city construction

  - A single hull is selected, and objects are attached to its outside
    - Windows
    - Doors
    - Lamps
    - Pipes
    - etc...
- Special information will be pre-defined for each object
  - Grabbable areas
    - Ledges
    - Knobs - handholds
- Pathfinding graphs are built automatically:
  - Interactive objects are connected by proximity
  - Near horizontal hull polys are converted to navigable areas
  - Each building will define an area in a hierarchical pathfinding graph

## Game Assets

The following is a list of all assets needed by the game. These will be loaded at runtime. [Actual list may vary and most of this is TBD]

### Models

1. Building Sets
    a. Bottom Floor Wall Side
    b. Bottom Floor Wall Corner
    c. Middle Floor Wall Side
    d. Middle Floor Wall Side
    e. Flat Roof Corner
    f. Flat Roof Side
    g. Flat Roof Middle
    h. Peaked Roof Angle End
    i. Peaked Roof Angle
    j. Peaked Roof Riser
    k. Peaked Roof Riser End

### Audio

#### *Music*

1. Happy Background Music
    a. Bass Layer
    b. Melody Layer
    c. Alternate Melody Layer
    d. Percussion Layer
2. Dark Background Music
    a. Bass, Melody, and Percussion Layers

#### *SFX*

1. Footstep
2. Jump
3. Assorted yells, grunts and breathing sounds
4. Grab
5. Victory
6. Start Chapter

### Art

See Appendix B.

## Manpower Allocation

5 Programmers

1 Producer

5 Artists

1 Composer/Musicians

## Technical Features

### Overview

The game engine is composed of eight sub-systems, these are: the Memory Manager, the Message System, the Graphics Engine, the Physics Engine, Scripting/Game Play Engine, the Sound Engine, and the Input Engine. The Message System is the backbone of the engine, everything communicates through it. We are also using a distributed component model for our game entities. This means that each engine has a list of the components it cares about and nothing more. This also means that where ever data needs to be shared, such as position, each engine has its own copy.

The goal of all of this is that no engine has to know about any other engine, with the exception that every engine must know about the Message System if it needs to communicate and the Memory Manager if it needs dynamic memory. The reason for the communication all going through one point and the data not being shared is for multithread safety. Multithreading will be an important part of the game. Making a thread task is very simple, and will be used to load in assets, or to perform calculations such as path finding while still playing the game. Multithreading is also relevant to the Graphics engine, which will be running on its own thread, to give more time to the CPU for the rest of the game. To avoid

Below is a short description of the Engines as well as an outline of what features we are planning for each one.

### Scripting

The Lua Engine is the core of our data driven design. All data needed for game objects or engine configuration is loaded from a Lua file, and can be reloaded at any time. This gives the ability to iterate over data without the need to recompile. Levels and level information is also loaded through Lua. So the Scripting engine will be responsible for sending messages to other engines about what entities to create and when, as well as when to destroy objects based on game play mechanics such.

Level

a. Lua script parsing – Load .lua files from hard disk and execute or store procedures.
b. Lua configuration scripts – Configurable values for game initialization, including startup parameters, user controls, and user settings.
c. In-Game Console –Press '~' in game to show console. User may run select commands or change values on the fly.

  d. Tweakable Console Values – Changeable values from the Lua Console.
  e. Lua Object Factory – Objects defined, loaded dynamically and initialized by Lua scripts.
  f. Lua Level Loading – All level data loaded from a Lua script.
  g. UI Scripting – GUI is defined and scripted using Lua

B. Level
  a. All gameplay code in Lua
  b. Lua controlling objects

### Graphics

The game's graphics engine will support multiple animations pre-generated from 3DSMax. It needs to be tightly linked to an IK (Inverse Kinematics) system. The IK system will give us the freedom to manipulate the limbs of the character to make him grab, look, and jump appropriately. On top of that the rendering will be made out of two major features. First, the shader system. It is meant to support multiple styles of rendering, one for the regular look of the game (textures and phong shading) to capture the Victorian style of the time, wireframe for debugging, and finally, some other special effects to polish the whole experience. The other part of the renderer is the particle system; meant to handle volumetric effects like steam, dust, or other system we might need later. We are using HLSL and SEMANTICS for the shader and the whole engine is written using the DirectX 9 API.

[ABC Levels TBD]

## Physics

The Physics-Controller pipeline is as follows: (1) Force-Accumulator (2) Physics Integrator (3) Broad-phase Collision Detection (4) Narrow-phase Collision Detection (5) Collision Resolution.

During the Force-Accumulator step, all forces in a single object's force list are summed together and then multiplied by one over the object's mass to obtain acceleration.

Inside the Physics Integrator, each object is treated as a particle system with distance constraints. Each constraint represents an edge of the object's bounding volume, where as each particle represents a vertex of the object's bounding volume. Integrating in this way means no torque or angular momentum is calculated. Instead, these physical properties of real-world objects are implicit in the constraints binding the particles together as a rigid body, and thus rotational physics comes for free within the simulation. Additionally, because the particle integration is done through the time-corrected Verlet method, velocity is also implicit. Therefore, each object simply tracks its own acceleration, current position, and previous position.

The Broad-phase Collision Detection portion of the pipeline uses an implicit Octree stored as a hash table of object lists. Each node within the Octree has a unique ID built from its spacial location. This ID is the key hashed to obtain the list of objects intersecting the octant (tree node). Another property of the Octree used in *Run!* that differs from the straight forward Octree implementation is that only leaf nodes store the list of objects intersecting the node. Objects do not get stuck in internal nodes. This was done because the Avatar and other dynamic objects within the game are relatively the same size as

Octree leaf nodes unlike the buildings of the city. Meaning that building will intersect and reside in many nodes, and dynamic objects will most likely not reside in multiple nodes. Static objects (buildings) are allowed to reside in multiple nodes as their spacial location within the Octree will never be recomputed. So, placing them in several nodes is inexpensive at design time.

Narrow-phase Collision Detection operates on the list of objects passed from the broad-phase. This list is further trimmed with sphere and AABB(axis-aligned bounding box) rejection/acceptance. Those objects accepted are then run through GJK(Gilbert-Johnson-Keethi). If successful, then the GJK output simplex is cached for each collision pair.

Collision Resolution uses the cached simplex for each pair to determine the minimum penetration depth for an impulse based resolution. The simplex can also be used to find a contact formation (aka contact manifold) to use as input into the Verlet Integrator next loop to simulate both linear and angular(rotational) motion.

A. Level
    a. **Bounding Volumes:**
        i. Bounding Sphere
        ii. AABB
        iii. Convex Set
    b. **Bounding Volume Hierarchies:**
        i. Collection of Bounding volumes to represent indivdual game entities
        ii. Used to augment animations and solidify them in a physically simulated steampunk/victorian environment
    c. **Collision Detection:**
        i. *bool* value reporting intersection between any two of the above volumes:
            1. Non-Convex Set vs Non-Convex Set uses optimized special case code
            2. Any intersection involving a Convex Set uses GJK (Gilbert-Johnson-Keethi)
    d. **Collision Resolution:**
        i. Non-Convex Set vs Non-Convex Set uses optimized special case code
        ii. Any intersection involving a Convex Set uses an Enhanced GJK (Gilbert-Johnson-Keethi) algorithm to build contact data:
            1. Contact data used to resolve interpenetration through Time-Corrected Verlet Integration
    e. **Collision Response:**
        i. Contact data used to generate Force, Impulse, Torque, and Impulsive torque
        ii. Impulse/Impulsive-Torque converted to linear/angular velocity
        iii. Linear/Angular velocity used to update positions
    f. **Accumulator:**
        i. Converts Force/Torque to linear/angular acceleration
    g. **Time-Corrected Verlet Integration:**
        i. Uses constraints of Convex Set
        ii. Uses current/previous position and linear/angular acceleration to update physical simulation of rigid bodies and particles

B. Level

a. **Cloth/Soft Body Simulation:**
    i. Used for canopies as trampoline surfaces
    ii. Takes advantage of constraints in Convex Set
    iii. Takes advantage of Time-Corrected Verlet Integration
    iv. Uses Triangle intersection tests and resolution

b. **Ragdoll:**
    i. Takes advantage of contraints in Convex Set
    ii. Takes advantage of Time-Corrected Verlet Integration

C. Level
  a. **Fluid Dynamics:**
    i. Used for steam dynamics

## Messaging

The Message System allows an engine to communicate with another without prior knowledge that the other engine exists. Messages use reference counted smart pointers, so engines don't have to worry about deleting allocated memory used by the message. If an engine wants to listen for a message it needs to register for the message type, which is a string. Once a string is registered it will be associated with a unique ID. This way message can be compared by an unsigned rather than a string. When and engine registers for a message it also gives the address of the member function it wants to use to respond to that message. Then in the engines Run() function all it needs to do is call the HandleMessages() function of its message receiver.

A. Level
  a. Register and Unregister for messages on the fly
  b. Communication between engine components
  c. Ability to send data to other engines.
  d. No need to register for a message you want to send
  e. Logging Messages to file

B. Level
  a. Debug info on messages such as who sent them and to whom

C. Level
  a. Load Messages from File for replay

## AI

The AI engine will use a navmesh to describe the level geometry. The navmesh will be constructed by simplifying game assets into convex hulls. Each convex polygon in the hull will describe an interactive surface to the avatar. The mesh data will then be used in conjunction with the user input to determine animation and movement for the avatar. An animation graph will be used to describe transitions between actions and animation cycles so that the avatar acts in a believable manner. Enemy AI agents will be spotlights and airships. These will exist primarily for atmosphere and will not require any major intelligence beyond simple avoidance and wandering behaviors. The camera will also use the navmesh graph to find points of interest along the avatar's current path. It will work to keep the upcoming action visible to the player.

A. Level
   a. Input intent prediction
      i. Combines input and the pathfinding graph to determine the most probable path that the player wishes to take
      ii. Attempts to avoid issues with missing jumps because of poor camera angles
         1. ex: User jumping towards camera
      iii. Attempts to avoid frustrating the player by not demanding excessively precise
         1. ex: Correcting the jump vector across a gap to a grabbable point if the input is a few degrees off
         2. ex: Jumping over an edge with no path nodes in sight will still cause the player to fall
   b. Intelligent camera:
      i. Evaluates player movement and tracks a point along the predicted path.
      ii. Attempts to keep a wide view of the action so that the player does not have to constantly move the camera to see what is going on
   c. Pathfinding graph:
      i. Defines areas of mobility in the level
   d. Animation control:
      i. Analyzes player motion and the environment to blend appropriate animations for a seamless and believable character

## Networking

A very low priority for Run!, but still worth mentioning is ability to log on to a server and upload best times, and high scores. This will be done via a PHP and MYSQL database driven system using http requests sent via win32 network sockets. Even though this isn't networked game play, TCP will be used to make the connection and insure that the data arrives. The data will be encrypted since users will be logging on. The goal will be to send secure data to take away the chance for users to have bogus scores on the boards.

## Coding Methods

Run! uses Doxygen to for documentation, and it is compiled weekly to make sure it is up to date. Classes and functions are named using camel case notation, where every new word is capitalized. Local variable and parameters use Hungarian notation, where the first letter is lower case then every word is capitalized. Member variable use the m_ notation, globals variables and static variables use g_ and s_ respectively. All defines and const values use all capitals and words are separated by underscores. Files names have a three letter prefix followed by an underscore based on what overall system they fit into, so all graphics files have gfx_ as the prefix. For source control Run! uses Tortoise SVN, and commits to SVN should include a description of what changes were made, or what files were added.

## Tools

The first tool made for Run! was the Class Creator. The Class Creator works as a plug-in for Visual Studio, where you can make a class, create members functions and variables and the application will write the necessary code with documentation.

Run! will also include an in game level editor, where you can place building pieces together like legos , to create a city. This will also include 3D picking to allow users to place or move objects anywhere in the world. Picked objects are the only pieces that can move, if picked object collides with a non-picked object it will stick to it. Once a city is build from the lego blocks, the editor will simplify the model by taking removing any vertices inside the building.

A 3D Studio Max exporter is being used for the models, that contains information such as convex hulls, bone weights, animation key frames, bone constraints such as length from joint to joint and angular thresholds.

# Appendix B: Art List

## Models

### Character Model

**Character:**

Aldus Traceur, Main Character

**File Name:** Character Model_NoRig

**Model Description:** Main Character model, male.

**File Format:** .max

**Polygon Count:** 5,167

**File Size:** 1,067 kb

**Animation Details:**

Currently we have a rigged and weighted file for animation. The animations are being saved as separate documents. The model is to be animated using biped and keyframes. The animations will be shorter parkour actions that can be later combined for more complex movements by the programmers. The idle and ready stance animations will provide strong transitioning elements into all action animations. The movement will be aided by the use of dummy objects. There is the capability of secondary motion for the belt and bag he is carrying.

Animations to date are run and jump. The next animations in line for December are the ready stance, idle, leap, speed vault, fall, hang, land and wall kick.

### Environment Models

**Buildings:**

Building Sets – 5+ total sets by Summer 2009

**File Name:** bs_a-n_{insert additional details here}

**Model Description:** Full set of pieces that can be combined to make a building.

**File Format:** .max

**Polygon Count:** to be determined

**File Size:** to be determined

**Building Set Details:**

Breakdown of building sets will be as follows.  Sets in total will be 15 pieces.

**Floor:**

Ground Floor

A. Ground Floor Corner Piece
B. Ground Floor Face Piece

Second Floor

C. Second Floor Corner Piece
D. Second Floor Face Piece

**Roof:**

Flat Roof -- Parapet

E. Flat Corner Piece
F. Flat Roof Edge Piece
G. Flat Roof Flat Piece

Tall Roof

H. Tall peak (high slope tip)
I. Short Peak (low slope tip)
J. Internal Cube for stacking
K. Short roof side
L. Taller Internal Cube for stacking
M. Tall peak side

**1 Door**

**1 Window**

**Pipes:**

Steampunk Pipes  -- 1 pipe completed to date.

**File Name:** pipe_2_txt

**Model Description:**  Corroded pipe to add ambiance to city.

**File Format:**  .max

**Polygon Count:**  86

**File Size:**  274 kb

**Details:** Corroded pipe with diffuse, normal and specular map applied.

**Chimneys:**

Flat rooftop Chimneys – 1 chimney completed to date.

**File Name:** SqCh_Final.max

**Model Description:** Chimney. This specific chimney is a brick chimney with soot.

**File Format:** .max

**Polygon Count:** 172

**File Size:** 252 kb

**Details:** Chimney with diffuse, normal and specular map applied.

**Additional Windows:**

Low Poly Windows – 1 completed to date, with 4 changeable window silhouettes.

**File Name:** Window1.max

**Model Description:** Low Poly Window made to be seen from a distance.

**File Format:** .max

**Polygon Count:** 14

**File Size:** 232 kb

**Details:** There are silhouettes that are swappable within the texture levels. Current silhouettes can show a figure in the window, writing desk, cat, or curtains.

**Steam vent:**

**File Name:** SteamPipe1.max

**Model Description:** Steam vent. This specific vent is a metallic vent pipe with a cap.

**File Format:** .max

**Polygon Count:** 32

**File Size:** 240 kb

**Details:** No additional details.

**Chairs:**

Rooftop Chair – 1 model.

**File Name:** rooftop chair LP wooden chair.max

**Model Description:** Someone forgot their old wooden chair out on the rooftop.  Oops.

**File Format:**  .max

**Polygon Count:**  108

**File Size:**  224 kb

**Details:** Model to date, needs texturing.

**Lanterns:**

Lanterns will be used as additional light source.

**File Name:** to be determined

**Model Description:** Hanging lanterns for sides of building or rooftop balcony.

**File Format:**  .max

**Polygon Count:**  to be determined, goal is no more than 100 polygons.

**File Size:**  to be determined

**Details:**  Metallic and cast-iron texturing will be used.

**Facades:**

Facades will include exterior wall embellishments and crown molding.

**File Name:** to be determined

**Model Description:** Visual elements that will tie in the Victorian feel.

**File Format:**  .max

**Polygon Count:**  to be determined, goal is no more than 100 polygons.

**File Size:**  to be determined

**Details:** Will be created using spline and/or box modeling.  The molding is planned to be tileable so it can be used on a variety of building types. Wall décor will be able to be placed individually wherever required.

**Gears:**

Gears will function as an additional visual asset. They can be placed around the radius of pipes, strewn about or used within other models to give it that steam punk quality. There will be 5 sizes and styles of gears that will be utilized. Interior radius will be the same as to allow for use on a specific dimension of object.

**File Name:** to be determined

**Model Description:** Gears, various sizes and side amounts.

**File Format:** .max

**Polygon Count:** to be determined, need to be low poly.

**File Size:** to be determined

**Details:** no additional details.

**Mayor's House:**

One of the specifically planned buildings. It will be a Victorian Era high society home.

**File Name:** to be determined

**Model Description:** Large home with an air of extravagance. Very traditional Victorian designs. Will need to fit with the design of the mayor and his daughter, who both live in the home. Steampunk elements will be present, but will be evident in shapes of the façade on the home.

**File Format:** .max

**Polygon Count:** to be determined, goal of no more than 1,500

**File Size:** to be determined

**Details:** One of the focal point buildings. Goal is for design to really aid to establish the setting of the game.

**Clock Tower:**

Second specifically planned building. Initial designs have been concepted.

**File Name:** to be determined

**Model Description:** Clock tower that is one of the tallest points in the city.

**File Format:** .max

**Polygon Count:** to be determined, goal of no more than 2,000.

**File Size:** to be determined.

**Details:** Tall clock tower that reads very clearly steampunk. Strong silhouette read. Concept is based on a pendulum type tower, where the clock and its gears hang off of a very large arch. Lots of gears and metal texturing.

**Blimp:**

Third specifically planned object.

**File Name:** to be determined

**Model Description:** Large blimp that will go across city with ladder hanging from underside.

**File Format:** .max

**Polygon Count:** goal is no more than 1000.

**File Size:** to be determined

**Details:** Large flying blimp. Very texture dependent for design.

**Steam Factory:**

Last specifically planned object.

**File Name:** to be determined.

**Model Description:** Very steampunk feel, with lots of gears and pipes present. Metal and brick texturing.

**File Format:** .max

**Polygon Count:** goal of no more than 1500.

**File Size:** to be determined.

**Details:** Large in width compared to other buildings. Initial concepts have been done to determine look of object.

## Textures

All game objects will be textured with at minimum a diffuse map. All textures will be saved as .jpeg, 32 color depth and will be 256x256 maps. High poly models will have both specular and normal maps present. The goal is to have diffuse, specular and normal maps on all objects that will come in close proximity to the character.

**Current Model Textures:**

**Pipe**

**File Name:** pipe_diff_256, pipe_nm_256, pipe_spec_256

**Texture Description:** Color, Normal and Specular map for pipe

**File Format:** .jpg

**Color Depth:** 32

**Chimney**

**File Name:** SqCh_256, SqCh_nm, SqCh_spec

**Texture Description:** Color, Normal and Specular map for chimney.

**File Format:** .jpg

**Color Depth:** 32

**Window**

**File Name:** Window 1 Diffuse, Window 1 NM

(Same for different silhouette textures, Window 2-5)

**Texture Description:** Color, Normal map

**File Format:** . jpg

**Color Depth:** 32

**Tileable textures:**

Two brick variants and 3 concrete variants for general use.

All are diffuse only currently. Textures are saved as .jpg at 32 color depth.

# Appendix D: Schedule

## Milestones

### Engine Proof

November 5th, 2008

The engine proof build of *Run!* was a basic run through of our current framework.  For graphics, this included the core rendering component of the game, including basic phong lighting, shader support, and mesh loading.  This also included the first iteration of our animation system.  Basic physics support was implemented using collision and response between basic primitives.  Basic player logic was implemented, with a chase camera behind a player running and jumping through a test scene.  Sound was also supported, with a sample jump sound effect and background music track.  Sample UI work was implemented to provide splash screens and pre-game menu.

### First Playable

December 1st, 2008

For first playable, we plan on having a sample run through of a game situation.  This will include a sample city area that the player can jump through, using an updated graphics component featuring upgraded lighting.  The city will be constructed using the basic first iteration of our in-engine level editor.  As needed, physics is ready to be expanded past to support the inclusion of grabbing and vaulting based moves.  The game logic will be updated to support grabable objects.  Using these, the player will be able to run through the city using the grabs and vaults.  The engine will also feature an expanded use of user interface mechanisms to provide for our comic book cut scenes.  Art and audio will continue as they have been, creating assets to be moved into the game as they are completed.

### Alpha

Early February, 2009

By alpha, we plan on having a relatively complete experience for the user.  Animation and graphics work will be iterated from their previous point to provide for smoother animations and a better overall feel for the world.  With the systems in place from the first playable, iterating on the gameplay will be a relatively easy process.  We will have the framework available to easily implement new story scenarios with their own cut scenes.  Also, with the work on the level editor being nearly complete, we will be able to quickly construct sections of the city for testing.

### Beta

Mid March, 2009

This build will be a purely iterative improvement over alpha.  The systems for the engine will be relatively complete, other than bug testing, prior to this build, so we will be able to focus on testing the product for the majority of this phase.  The big addition will be with regards to an expanded move set.

As the artists complete more animations, we will be able to implement their actions into the game in an efficient manner.

### Final

Late April, 2009

As with the Beta phase, this will be an iterative process.  Even more testing will be done to bring our game to gold status.  Using this feedback, we will be able to quickly iterate through different gameplay scenarios to figure out which ones work the best.  Final bug testing and engine tweaking will provide with the final engine build.

# Appendix E: Risk Analysis

## Major Risks

- Lack of Time
  - Risk: Due to our decision to rewrite the core architecture, we have run dangerously low on time to bring the game to a completed state. We have less than two months between alpha and final to complete a playable product.
  - Mitigation: There's not too much we can do to control time. At this point, we'll be putting in a lot of hours to bring the engine back together so we can work on gameplay. From there, we'll have the team working in two groups, one on the editor, and one on the game. With just a few weeks remaining, we'll be able to move to a phase of development where we have heavy testing multiple times a week, and programmers devoted solely to working on gameplay code. With the editor ready for use, we will also be able to quickly iterate on the level design as well.
  - Mitigation: We also have a large amount of art assets already completed by the artists. Due to this, we have a large backlog of assets ready to integrate directly into the game. The technology is already in place to import models and animations directly into the engine, and we will be able to quickly test new assets as they are completed.
- Game Scope
  - Risk: Simply put, our game is huge. To bring a steampunk-victorian city to life, we need a lot of art. To bring the art of parkour to life, we need a lot of animations. These things alone would be a ton of work, but together they represent a huge hurdle for our game.
  - Mitigation: The art list has been prioritized to get us the most important art first. This includes just a couple of building sets to construct the world, as well as the basic move set for Parkour movements. This will allow us to still construct our world and character movements, while shifting the low priority work to the end of the project.
- Fun Factor
  - Risk: Although we are confident that the game idea will be fun, at this point in the project, we have no way to gauge the level of fun that the end product will actually have.
  - Mitigation: Testing Testing Testing. As the game progresses in the development cycle, we will have to ramp up the testing process. As this continues, we will need to make sure that we continue to listen to the feedback that our testing base provides in order to make the best game that is possible.

## Appendix F: Copyrights

- All art used in *Run!* has been created by our art team with the purposes of being used in-game.
- Assassin's Creed ©2008 Ubisoft Entertainment
- Tomb Raider ©Eidos Interactive Limited. 2008

# Appendix G: Team

## Programmers

1. Matt Casanova – Engine Architect, Technical Director
2. Mike McCullough – AI Lead, Gameplay programmer
3. Serge Metral – Graphics Lead, Testing Director
4. Ryan Edgemon – Physics Lead
5. Randy Knapp – Game Designer, Gameplay programmer

## Producer

6. Dan Weiss – Lead Producer

## Artists

7. Marisa Brown – Art Director, Environment Artist
8. Matt Broderick – Animation Lead, Character Modeler
9. Ashley Calkins – Modeler and Texture Artist
10. David La – Lead Character Design, Concept Artist
11. Josh Jones – Character Animation
12. Creighton Medsker – Modeler and Texture Artist
13. Tyler Woods – Digital Sculptor, Musician

## Composer

14. Tyler Glaiel – Composer, Musician