# You Say Jump, I Say How High? Operationalising the Game Feel of Jumping

**Martin Fasterholdt, Martin Pichlmair**
ITU Copenhagen
Rued Langgaards Vej 7
2300 København S
MFasterholdt@gmail.com, mpic@itu.dk

**Christoffer Holmgård**
NYU Tandon School of Engineering
6 MetroTech Center
Brooklyn, NY 11201
holmgard@nyu.edu

## ABSTRACT

This paper explores the design of jumping in 2D platform games. Through creating a method for measuring existing games, applying this method to a selection of different platformer games, and analysing the results, the paper arrives at a comprehensive data model for jumping. The model supports the exploration, design and development of new jump implementations. The underlying framework and toolset can be used by game designers to measure, model and analyse movement in platform games.

## KEYWORDS

game feel, game design, jumping, game input, modelling & simulation

## INTRODUCTION

When designing and implementing games, developers often talk about the game feel of a particular mechanic or other aspects of a game. Swink defines game feel as "Real-time control of virtual objects in a simulated space, with interactions emphasised by polish." (Swink 2008, p. 6). This paper answers the question of how the game feel of jumping is modelled in 2D platform games by measuring and modelling the exact features involved. In the following paper we first give an overview of related work on jumping in games followed by existing methods of examining jumping in platform games. This forms the starting point for specifying a method and constructing the required technical tools to research a number of 2D platform games, specifically *Super Mario Bros. 3* (Nintendo 1988), *Super Meat Boy* (Team Meat 2010), and *Limbo* (Playdead 2011). We use the results to identify design patterns in the jumping mechanics in these games and analyse how these patterns influence game feel. The presented framework and toolset is intended as a starting point for further formalisation of movement in games and for future experiments.

## RELATED WORK

The following section provides a brief history of jumping in games together with an overview of existing analytical approaches that served as inspiration for the method we developed.

**Proceedings of 1st International Joint Conference of DiGRA and FDG**

## The History of Examining Jumping in Games

Butler (2014) provides an overview of jumping as a central mechanic through the history of video games. This overview begins with *Donkey Kong* (Nintendo 1981), which established the platform game genre and was one of the earliest games to feature a jumping character. *Super Mario Bros.* (Nintendo 1985) expands on this jump and is cited as the first game allowing the player to control the height and distance of the jump. Since then, jumping has been used in countless games. Some of them greatly expand on the basic jump. One example is *Braid* (Number None 2009), where the player can manipulate the flow of time while jumping through a series of puzzles. Jumping continues to be an essential part of today's 2D platform games. Two examples of recently released jump-centric games are *Ori and The Blind Forest* (Moon Studios 2015) and *Feist* (Bits & Beasts 2015).

Another approach is to look at how parameters of jumping have developed over time. This can be seen in the work of Lefky and Gindin (2007), who explores how gravity has developed over time in a number of games of the *Mario* series. According to their research the force of gravity has consistently been reduced as new games have come out. It appears to be converging towards the force of gravity on Earth. *Mario*'s gravity is, however, still significantly stronger than Earth's gravity. Another example is Thompson (2015), who explores how design patterns have been reused and evolved across different *Mario* games. Begy (2010) on the other hand has examined the wider cultural meaning and prevalence of jumping in games, analogue and digital. His findings and analyses suggest that the appeal of the feeling of jumping comes from the jump being a metaphor for empowerment, domination, and overcoming. Warren (2014) suggests that this feeling is amplified in games where characters have jumping capabilities far exceeding those of humans.

## Jump Feel

Swink (2008) defines three different constituent components of game feel: Real-Time Control, Spatial Simulation and Polish. All three contribute to the game feel of a jump. While Swink's framework is universal, a number of authors have looked specifically at the feel of jumping in games. For example, Rogers (2010) argues that Mario owes most of his fame to sticky friction, which in short is Mario's inertia and feeling of weight. Rogers (2009) also examines how the significance of jumping has developed over time and compares the feel resulting from different jumping implementations. Swink (2008) makes a number of case studies where he analyses the jump and resulting game feel in a number of games including *Super Mario Bros.* and *Bionic Commando* (Capcom 1988). These and similar analyses provide valuable reflections on game design.

This paper brings a new perspective to the existing literature on jumping in games by suggesting that game feel can be understood by examining jumping and the game world parametrically. The analysis presented in this paper will discuss the game feel of different games, but arguments and claims will be based on measurements of movement rather than intuition and self-report. This will yield information about how each part of the implementation influences game feel. Before describing our own method for understanding the constituents of jump feel, we will briefly relate three empirical approaches that serve as inspiration for the work presented here: Understanding jump feel from implementation, memory inspection, and observation of timing.

## Jump Feel from Implementation

One way of understanding how a particular kind of jump feel is achieved is examining specific implementations of jumping through the eyes of their designers and developers. Miyamoto and Tezuka explain that they designed the first level of Mario to feel enjoyable while gradually teaching the player about the game. They also briefly touch upon how the movement was designed to give Mario a feeling of weight (Eurogamer 2015). Other developers have gone to great lengths to explain their implementations of jumping and how they overcame design challenges. Saltsman (2010) has described in great detail how he tuned his game *Canabalt* (Semi Secret Software 2009). He specifically talks about the camera perspective in the game and the run speed of the character. Similarly D'Angelo (2015) has written an in-depth article about the movement of the Plague Knight character designed for the expansion to the game *Shovel Knight* (Yacht Club Games 2014). The article focuses on how the movement of the character was designed to feel wild, maniacal, and explosive while being compatible with the level design of the original game. Another example comes from the online extras to *Indie Game: The Movie* (Pajot and Swirsky 2012). Here the developers of *Super Meat Boy* are interviewed and developer Refenes explains how they noticed players failing to do wall jumps. He dealt with this by adding a delay of about 1/20 of a second when the player navigates away from a wall slide. All these discussions of implementation specifics are anecdotal and only focussed on specific details of the jump.

On the other hand there are many articles about the technical implementation of a simple jump. Monteiro (2012), in his guide to implementing 2D platformers, presents a variety of different ways to implement a platform game. The guide begins with the purely tile-based approach seen in *Flashback* (Delphine Software 1992) and *Prince of Persia* (Brøderbund 1989) and ends with a vector-based approach based on physics engines seen in games like *Braid* or *Limbo*. Other sources for information about the game feel of jumping are tutorials which describe the implementation process step by step. Tutorials usually also provide the source code free of charge. Pignole (2013; 2014) has published several of these, some specifically about 2D platformers. One of his tutorials focuses on collision between the character and the environment and presents a solution using ray-casting (Pignole 2013). Another of his tutorials focuses on implementing movement and jumping, that feels neither limp nor rigid. He also goes through the implementation explaining how different elements will influence the movement of the character (Pignole 2014). Venturelli (2014) has written a similar series of tutorials on the topic. In Game Feel Tips II (Venturelli 2014b) he focuses on how the implementation of speed, gravity and friction influence the movement of the character. He expands on this in Game Feel Tips III (Venturelli 2014c), where he also provides playable examples. These articles provide a good foundation for implementing jumping, but generally lack detail. They may be useful for implementing a jump mechanic with the desired jump feel, especially when the game is close to an existing game, but a more general approach is needed to permit designers to explore instances of jump feel that diverge from existing templates and allow room for exploration and experimentation.

## Jump Feel from Memory Inspection

How can movement in games be measured? One way is to look at the computer memory used by the game while it is running. By inspecting and experimenting with different parts of the memory it is possible to find the exact values used by the game, for example the

acceleration of the character when they start to run. This approach is especially suited for researching older games. They use far less memory than modern games so locating the correct memory values is easier. Aldrich (2012a-2012d) used this method to gather a large variety of the parameters used in a range of *Mario* games. His research includes *Super Mario Bros.* (Nintendo 1985), *Super Mario Bros. 3* (Nintendo 1988), *Super Mario World* (Nintendo 1990), and *Super Mario Advance 4* (Nintendo 2003). A similar approach has been applied to *Sonic the Hedgehog* (Sonic Team 1991) resulting in a very detailed guide describing the physics of the game (Sonic Retro 2014).

A comparable approach more suitable for modern games is to use a debugger to inspect the game while it is running. Modern games use a lot more memory and the debugger provides search functionality, which makes locating the correct values more approachable. This method has been applied by Olney (2013) who used the *OllyDbg* debugger as well as an unspecified memory searcher to inspect *Super Meat Boy*. The result is a breakdown of how the basis of the game could be implemented. He shares his implementation, but omits the specific values of the parameters. These methods result in very detailed measurements and are highly relevant to this project, especially the measurements of *Mario* which will be used throughout the project. However, applying them requires extensive technical knowledge and is extremely time consuming.

### Jump Feel from Timing

A more efficient, but far less accurate, way of measuring movement in games is to use a stop watch. Daniels (2013) employs this method to compare the jump duration between 2D and 3D games. His way of measuring the jump duration is to start a stopwatch at the same time he presses the jump button, and to stop it when the feet of the character touches the ground. He concludes that what he considers "good feeling jumps" have roughly the same duration in 2D and 3D games, lasting somewhere between 0.7 and 0.8 seconds. These kinds of experiments are easy to conduct but imprecise. However, the idea of timing the movement appears promising. A more accurate approach is described by West (2013). His method measures the delay between physical input on a controller and visible reaction on the screen. Briefly explained, he uses a video camera to simultaneously record a controller being pressed and the resulting reaction on a TV screen. The recording can be inspected, and the amount of frames between input and response counted. Knowing that the camera records at 60 frames per second, he can convert frames to seconds which gives him his measurement. This response time is an important factor for how it feels to interact with a game. Instead of focusing on response time, this project will primarily explore how input is translated to movement within the game world.

## METHOD

The method we present does not directly use any of the three approaches described above, but is similarly empirical. By measuring features of jumping in a number of specific games, we identify components of jump feel and show both, how they vary across the specific games, and how these features can be modified by parameters to create different kinds of jump feel. For each game examined we identify features and describe values, value ranges or parameters yielding a general jumping model.

## The Tools

The measurements were done using four tools, an input plotter, an input simulator, screen recording software, and a measurement tool for the recorded videos.

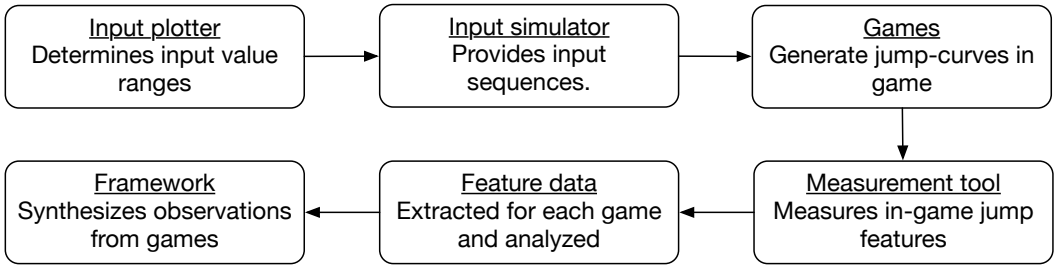



Figure 1: An overview of how the developed tools were used to generate data leading to the creation of the framework.

**The Input Plotter** was created in the *Unity* game engine to measure stick input. Measurements are taken by manually moving the stick around for several minutes while the tool samples the thumbstick about 60 times each second. The result is a visual representation of all values sampled during the measurement period. This provides fundamental understanding of the input received from the controller, which was required before we could start simulating input.

**The Input Simulator** was created to provide controllable, replicable and precise input to the game. The software mimics an XBox controller and sends analogue thumbstick and button input to the game. It is a C# Form Application that uses the EasyHook framework to inject DLLs and intercept API calls from the game to the operating system. The sample project and articles by Stenning (2010; 2011) formed the starting point for this application.

**Screen Recording Software** was used to record the resulting movement in high resolution and high frame rate. GeForce ShadowPlay was used for this project, it offers a GPU-accelerated video encoder that has minimum impact on CPU performance. Using this software resulted in recordings with a resolution of 1920x1080 with a steady frame rate of 60 frames per second.

**The Measurement Tool** was developed to facilitate the task of measuring the position of the character for each frame of the recording. The implementation is based on OpenFrameworks, an open source C++ toolkit which offers easy access to basic video handling. The program allows the user to create a list of character positions over time. The user simply clicks at the same reference point on the character for every frame. If the camera moves, an anchor point in the environment has to be selected as well. The program then considers all the points and calculates the position of the character in relation to the anchor point. The measurements are then exported for analysis.

## Measurement Process

Using the above tools, sequences of inputs were created for *Limbo* and *Super Meat Boy*. The sequence of input required was determined experimentally and iteratively for each game.

The result from each measurement was used to inform and refine the proceeding experiments. Measuring the duration of a basic jump was for example required before measuring the release of jump input.

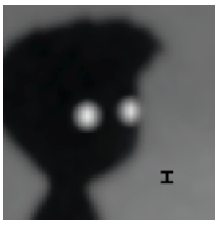
For each experiment the resulting movement was recorded and measured, except for *Mario*, where the data was taken from Aldrich (2012; 2012; 2012; 2012). All data was normalised in relation to the height of the character. This normalisation allows comparisons between games and negates effects of the camera zooming. The exactness of the measurement is limited by animations and dithering. Animation can lead to a disjoint between a character's physical and visual position. In an optimal case, the position of the centre of mass would be measured. Since this is not visualised, we picked a significant point as a reference point for the characters in each game (e.g. the eye of the boy for *Limbo*). Dithering is the imprecision introduced by averaging the color of a pixel based on adjacent pixels, when the character's position is not aligned with the pixel grid. This problem was limited by running the games at a high resolution.

For each game, a model was fitted to the measured data points by hand. The Root Mean Square Deviation (Maxwell and Delaney 2004) as well as the weak Fréchet distance (Campbell et al. 2015) for each of the models was calculated to find out how closely the model replicates the jump of the character. The results are shown in Table 1. Both of these methods are scale dependent, which in this context means faster movement is expected to create larger deviations.

Table 1: RMSD and Fréchet distance for two examples from *Super Meat Boy*.

| | **RMSD Distance** | **RMSD Height** | **Fréchet distance** |
|---|---|---|---|
| Super Meat Boy, Jump | 0.109 | 0.199 | 0.239 |
| Super Meat Boy, Running | 0.095 | – | 0.160 |
| Limbo, Jump | 0.025 | 0.034 | 0.053 |
| Limbo Running | 0.039 | – | 0.018 |

(a) Fréchet distance for *Super Meat Boy*



(b) Fréchet distance for *Limbo*.

Figure 2: Fréchet deviation in relation to in-game character size.

## THE MODEL

Based on the results of the experiments, a movement framework was defined which covers 21 basic features of movement and jumping in platform games. These are further divided into six categories as seen in (Table 2). Each feature is described using one or several values

Table 2: The 21 features used to describe jumps across the studied games.

| Category | Feature | Category | Feature |
|---|---|---|---|
| **Input** | Overshooting | **Air Control** | Maximum Air Speed |
| | Deadzone | | Air Acceleration |
| | Blowback | | Air Friction |
| **Ground Movement** | Maximum Ground Speed | | Air Turn Acceleration |
| | Acceleration | | Releasing Horisontal Input |
| | Deceleration | **Jump Release** | Minimum Jump Duration |
| | Turn Acceleration | | Additive Jump Force |
| **Jump** | Gravity | | Release Drag |
| | Terminal Velocity | **Details** | Jump Cache |
| | Takeoff Velocity | | Edge Jump |
| | Horizontal Takeoff Velocity | | |

and also describes factors directly influencing the feature as well as any special conditions. The choice of features was done with the aim of modelling the basic jump in all three games as adequately as possible. Not all features are present in all games. It is noted in the below paragraphs if a feature is specific to one game and can not be found in another.

The framework is not complete and is intended to form a foundation. It excludes a range of special jumps such as wall jumps, double jumps, hovering jumps, and back-flips. Furthermore, it does not cover movement on sloped terrain. The most common feature left out is the ability to sprint. These and many more features could be added to the model using the tools presented in this paper. *Super Meat Boy*, *Limbo* and *Mario* will be the primary examples. When needed, properties from other games are measured and included as examples to contrast the three case studies.

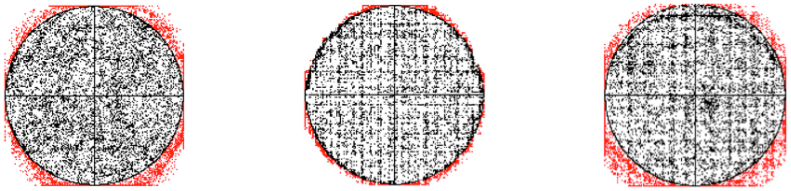
As previously mentioned, measurements are converted into units of character height to make them comparable across games. For example, if a distance measurement in pixels is equivalent to three times the characters height that measurement would correspond to 3 units.

## Input

While not unique to platform games, a detailed measurement of the input is relevant specifically for this kind of game because of the high level of precision and timing demanded from the player. Using the Input Plotter, a number of aspects of controllers were measured. Each measurement was performed by manually moving the left thumbstick, trying to cover the whole input space. This way, the input space as well as the outer maxima could be measured. Three different gamepads, an Xbox One, a PS4 and an Xbox 360 were tested for overshooting, deadzone and blowback. The input vector of a thumbstick consists of two values in the range of -1 to 1, the first value corresponds to the x axis (horizontal input) and the second to the y axis (vertical input). This expected limit is however not strictly imposed by the physical joystick as becomes clear below.

### *Overshooting*

When a stick is moved to its limit, the input vector received from the controller is expected to have a length of 1. In practice, this is not the case. Instead, all controllers tested occasionally



(a) Xbox One (b) PS4 (c) Xbox 360

Figure 3: Overshooting input visualised for the three controllers.

overshoot and register values that exceed this expected limit. Figure 3 shows the resulting values. Note that the individual axes of the input vector never go beyond the length of 1. In order to compensate for the overshoot, it is recommended to clamp the input vector to a length of 1. This will avoid irregular behaviour when using the length of the input vector as a factor for controlling the game.

### *Deadzone*

A Deadzone is a way to ignore input under a given size. When the player is not pushing the stick, the input vector is expected to have zero length. However, in practice, controllers sometimes output a non-zero input without being touched, since the stick is not resting perfectly in the centre position. The deadzone also influences the sensitivity of the thumbstick, as small movements of the stick are ignored. This is important when resting the thumb on the controller without the intention of moving it. Hesselgren (2012) and Sutphin (2013) provide comprehensive introductions to different deadzone implementations. The three examined games all feature circular deadzones. A circular deadzone ignores input vectors below a certain length. The cut-off length of *Limbo* is 0.358. *Super Meat Boy* has a deadzone value of 0.5. *Mario* was launched on a platform that did not support analog input and hence the concept does not apply. Four different deadzones can be seen in Figure 4.

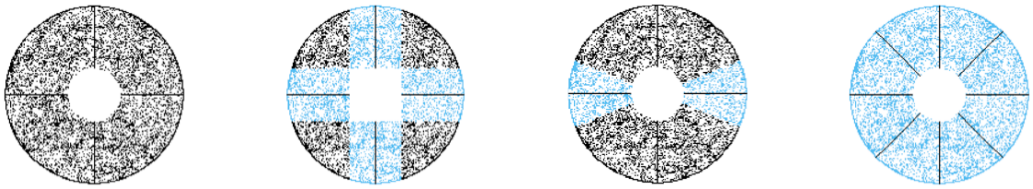
### *Blowback*

When a thumbstick is moved and then released, a spring pulls it back to its default centre position. Sometimes the stick will go fast enough to bounce briefly past the centre before settling, resulting in a blowback. The phenomenon should be considered when handling controller input, to avoid unwanted character movement. Consider, for example, the following very common case. A player is running right and decides to stop by removing the finger from the thumbstick. The stick is now pulled back by the spring and causes blowback. If the input resulting from the blowback is not within the deadzone, and the game is implemented to be as responsive as possible, the character will start going left. Blowback was measured in four directions: up, down, left, and right. An overview of the measurements for each controller can be seen in Table 3. This phenomenon was, to the best of our knowledge, formally described for the first time during this project.

Based on this limited data, blowback appears to be worth considering and testing for when developing a game. Using a deadzone higher than 0.777, which would be necessary to ignore the blowback input measured in this experiment, is not a viable solution, as this would drastically limit the input area available. We expect that solutions for this problem has to

Table 3: Input characteristics for three game controllers.

| | **Xbox One** | **PS4** | **Xbox 360** |
|---|---|---|---|
| Max overshoot | 1.264 | 1.100 | 1.277 |
| Max input at rest (X,Y) | $(0, 0)$ | $(0, 0)$ | $(0, -0.053)$ |
| Blowback Horizontal (min, max) | $(-0.279, 0.777)$ | $(-0.458, 0.264)$ | $(-0.190, 0.198)$ |
| Blowback Vertical (min, max) | $(-0.634, 0.438)$ | $(-0.264, 0.429)$ | $(-0.134, 0.000)$ |



(a) Circular Deadzone. All directions treated equally.

(b) Square Deadzone. Input snaps to horizontal and vertical axes.

(c) Bowtie Deadzone. Ideal for horizontal sweeps, e.g. when aiming.

(d) Directional Deadzone. Simulates a classic 8 way arcade controller.

Figure 4: Examples of four different kinds of deadzones. Input in the blue areas is adjusted before being accepted.

be developed specifically for each game's input requirements. The reaction to blowback was tested in *Super Meat Boy*, *Limbo*, *Braid* and *Spelunky* (Mossmouth 2013) and none of them featured such an implementation. The test consisted of running left and releasing the thumbstick. If blowback is not handled, the character is expected to occasionally flip and face right. This turned out to be the case for all four games. Additionally, in our limited tests, modern controllers had more blowback than older models, making the problem even more relevant on next generation consoles.

An additional observation was that in general all controllers exhibit a tendency to subtly change input values close to 0 on either the X and Y axis to an exact zero value on that axis, creating a snapping effect on the axis in question (independent of the other axis). This is observable in Figure 3, but was not tested further.

## Ground Movement

When the stick is pressed horizontally, the player character moves on the ground. The character accelerates from standing still to moving with a maximum speed.

*Super Meat Boy* and *Mario* both have just one acceleration and one maximum speed. In other words, the amount of horizontal input is irrelevant as long as it exceeds the deadzone. However, in *Limbo* maximum speed is dependent on horizontal input. Furthermore, instead of an acceleration force, the game appears to use animation transitions with fixed durations.

Mario has a deceleration that lets him come to rest after 0.457s. In contrast, Meat Boy instantly breaks when the stick enters the dead zone. Like with acceleration, *Limbo* appears to use a fixed duration when decelerating and coming to a stop takes 0.06-0.2 seconds.

### *Turn Acceleration*

When the player pushes the stick in one direction and then abruptly changes direction, the character decelerates, turns, and accelerates in the opposite direction. When this happens, turn acceleration is applied instead of the regular acceleration. This acceleration is applied as long as the input direction is opposite to the characters current velocity. Higher turn acceleration will result in a faster, more responsive, change of direction. Turn acceleration is the defining factor for Mario's iconic skid.

## Jumping

A jump consists of a takeoff, air time and a landing. The force applied to jump is the takeoff velocity. While typically just a vertical force, *Limbo* also adds a horizontal force at takeoff, making the character move forward as well as up.

All measured games handle takeoff, air time and landing differently. *Super Meat Boy* features a uniform vertical takeoff velocity of 21 u/s independent of the character's speed. *Mario* features a tiered jump with 4 different vertical takeoff velocities (13.75, 14.25, 14.75, 15.75 u/s) based on Mario's run speed (0-4, 4-8, 8-12, over 12 u/s) at takeoff.

*Limbo* features a linear mapping between vertical takeoff velocity and running speed. If the character has moved less than 0.05 units before the jump, the resulting jump is considered short, resulting in a 0.1 u/s penalty on the takeoff velocity. If the character has moved further before jumping, the resulting takeoff velocity is linearly mapped to 1.78-2.47 u/s. Since maximum walking speed is reached after 0.2 seconds in *Limbo*, the top takeoff speed is reached after that time, too. The lower takeoff velocities are therefore mainly relevant when partially pushing the thumbstick, resulting in a consistently slow walk speed. As mentioned, *Limbo* adds a horizontal velocity when jumping. This horizontal velocity is always higher than the vertical velocity, it is also not influence by walking speed and is consistently 3.5 u/s.

### *Air Movement*

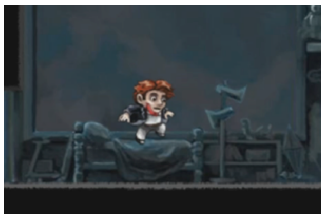
Once in the air, the character is subject to gravity. The differences in magnitude of gravity between the examined games are significant. *Mario* has a gravity of 75 u/s $^2$, *Super Meat Boy* has 41 u/s$^2$ and *Limbo* just 6 u/s$^2$. Considering the assumed body height of the characters, *Limbo* is the only game to have earth-like gravity. Gravity is applied as expected to Meat Boy and the boy in *Limbo*. *Mario*, on the other hand. has a terminal velocity of 17.25 u/s, after which the character stops accelerating vertically. *Limbo* is the only game of the three featuring air friction. *Mario* is the only game of the three where jumping is supported via a consistent additive upwards force. This force is applied as long as the character is moving upwards with more than 8 u/s and the jump button is pressed. The upwards force is smaller than gravity, which means the total vertical speed of the character is decreasing during the jump. As mentioned above, *Limbo* adds a horizontal force upon jump. No further horizontal acceleration is possible during the jump. *Mario* and *Super Meat Boy* on the other hand feature horizontal acceleration while the character is in the air. In both games, the characters accelerate similar to their ground acceleration while in the air. They also have the same maximum horizontal speed in the air as on the ground. Even their turn acceleration is the same as on the ground.

### *Landing*

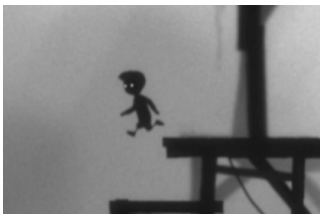
Fineberg (2015) presents various techniques for cutting a jump short. This effect is normally achieved by releasing jump input. Doing so in *Super Meat Boy* sets the character's vertical speed to zero, resulting in an instant vertical brake. The horizontal speed is not influenced. If the jump button is only pressed very briefly, resulting in an immediate break off of the jump, Meat Boy acts as if the buttons was pressed for 0.1 seconds. This results in a fixed minimum jump duration, a feature not present in any of the other games. When the jump button is released in *Mario*, the additive upwards force is no longer applied. As a result, gravity gradually forces Mario down, leading to a curved top of the jump curve. Releasing the jump button in *Limbo* results in the application of a vertical downwards force as long as the character is moving upwards. Additionally releasing the horizontal input increases the horizontal air friction to 9 u/s$^2$. This means that to achieve the largest possible jump the player will have to hold both jump input and horizontal input during the entire jump. Table 4 shows all the values identified across the three games.

Table 4: Movement values across the three games

| **Ground Movement** | **Meat Boy** | **Mario** | **Limbo** |
|---|---|---|---|
| Max Ground Speed u/s | 18 | 6 | $0.226 - 2.1$ |
| Acceleration u/s$^2$ | 35 | 13.125 | – |
| Acceleration Duration s | 0.516 | 0.457 | $0.06 - 0.2$ |
| Deceleration u/s$^2$ | – | 13.125 | – |
| Deceleration Duration s | 0.016 | 0.457 | $0.06 - 0.2$ |
| Turn Acceleration u/s$^2$ | 70 | 30 | – |
| Skid Distance u | 2.157 | 0.458 | 0.118 |
| Turn Duration s | 0.782 | 0.657 | 0.23 |
| **Jumping** | | | |
| Gravity u/s$^2$ | 41 | 75 | 6 |
| Terminal Velocity u/s | – | 17.25 | – |
| Takeoff Velocity Vertical u/s | 21 | $13.75 - 15.75$ | $1.78 - 2.47$ |
| Takeoff Velocity Horizontal u/s | – | – | 3.5 |
| Maximum Jump Height u | 5.554 | 5.287 | 0.529 |
| Maximum Jump Distance u | 18.599 | 5.7 | 2.037 |
| Maximum Jump Duration s | 1.05 | 0.967 | 0.85 |
| **Air Movement** | | | |
| Air Acceleration u/s$^2$ | 35 | 13.125 | – |
| Horizontal Air Friction u/s$^2$ | – | – | 6 |
| Air Turn Acceleration u/s$^2$ | 70 | 30 | – |
| Maximum Horizontal Air Speed u/s | 18 | 6 | – |
| Minimum Horizontal Air Speed u/s | – | – | $1.5 - 2.284$ |
| **Landing** | | | |
| Instant Break | yes | no | no |
| Minimum Jump Duration s | 0.1 | - | - |
| Additive Jump Force u/s$^2$ | - | 60 | - |
| Additive Jump Force, Threshold u/s$^2$ | - | 8 | - |
| Release Drag u/s$^2$ | - | - | 3.75 |
| Hold Jump Input | no | no | yes |
| Minimum Jump Height u | 2.211 | 1.475 | 0.392 |
| Minimum Jump Duration s | 0.467 | 0.4 | 0.65 |



(a) In *Braid* a jump input this far from the ground is stored and the jump performed when the character lands.



(b) Because of the edge jump in *Limbo* the character can still jump even though he is no longer touching the platform.

Figure 5: Subtle implementation details found while measuring the jumps.

## Details

There are a number of intricate details specific to the implementation of each jump in the discussed games. *Mario*, for example, is the only game of the three to feature a jump cache. If the jump button is pressed 1-2 frames before the landing, a new jump is triggered as soon as Mario touches the ground. A more noticeable example of this detail is *Braid*, where jump input is cached for 0.23 seconds. Meat Boy ignores any new jumps triggered while a jump is still in progress. Both *Mario* and *Super Meat Boy* require jump input to be released and repressed to perform a new jump. *Limbo*, on the other hand triggers a new jump as long as the jump button is held down. There is a delay of about 0.05 seconds after each landing before a new jump is triggered. This short ground movement results in the previously mentioned short jump. Therefore, the fastest way to move forwards in *Limbo* is to jump as much as possible, while making sure to run far enough between the jumps to avoid the short jump. Another detail in *Limbo* is a special delay after running over an edge where jumping is still possible, even though the character has already begun to fall. The player has up to 0.08 seconds to perform this edge jump successfully. This feature is called Ghost Jump by Venturelli (2014) and presented as a time delay by Pignole (2013). There are many more details to *Limbo* and other games that could be explored and documented in future work.

## CONCLUSIONS

The presented features all add to the specific feel of each game in question. When looking at the three presented implementations of jumping, it becomes clear that each of them is suited for the specific games it was made for. Mario's jump, for example, aids in stomping, i.e. landing vertically on enemies. This mechanic, that is common throughout the game, is supported by the relatively low horizontal speed, which reduces the amount of possible landing positions, making it easier to hit a target.

An iconic feature of Mario is his skid, the way he continues to move in one direction for 0.458u when the stick has already been moved to the other direction. The skid lends him weight and character. It also adds an element of danger to horizontal movement that further incentivises jumping, the core of the gameplay.

*Super Meat Boy* features a lot of deadly traps on the ground and mounted to the ceiling. Accordingly, the whole movement model of this game is geared towards avoidance of traps.

The instant vertical break is accompanied by a skidding-free horizontal instant brake. All of these features allow for very exact control of the character, which is balanced by an immense skid if the player does a turn instead of a break. It thus emphasizes mastery of the controls by punishing mistakes.

*Limbo* on the other hand is the slowest game of all three and this ties in with the game's moody setting. The game features the most realistic jump. If the boy is assumed to be 1.5m tall, his walking speed would be 3.15m/s, a quite realistic value. His highest jump would be 0.914m high, which is high, but still possible. Gravity in *Limbo* is around 9 $m/s^2$ – quite close to the 9.81 $m/s^2$ on Earth.

This research provides actual data points about jumping in platformers as well as a framework that links the relevant features together. It formalizes the results of design processes in order to support the exploration of new game styles. The jump features in the three games analysed support their core game mechanics and aesthetics.

## OUTLOOK

As the last step of this project, a jump prototyping tool was created in the *Unity* game engine. The tool contains implementations of the movement described in the framework and can simulate the three analysed games while exposing their features. Jumps can not only be performed in this tool, they can also be visualized as curves that are drawn and updated continuously, reacting in real-time to parameter changes and adjustments of the environment. The tool was not formally evaluated but it can be used to test the effect of changing different features of a jump. If the tool was polished to a higher degree, it would support scenario-testing to determine whether a specific jump fits the aesthetics aimed for by the designer. As of yet the tool is work in progress.

The logical next step in the development of the framework itself is to incorporate additional movement features. Sprinting, wall jumps, and double jumps would be obvious expansions. At the same time, similar frameworks for other types of games can be developed. Fighting games have enough similarities to be mapped like this. So do 3D games like First Person Shooters. The goal would not be to create a complete model of all possible movements in games, but to create functional prototypes to test specific move sets for their implications on the level design.

Another way forward would be to procedurally generate movement styles, that are tested against a designed level or generated level. Similar to Isaksen et al. (2015), the full parametrization of movement could form a basis for exploring the design space of a game procedurally.

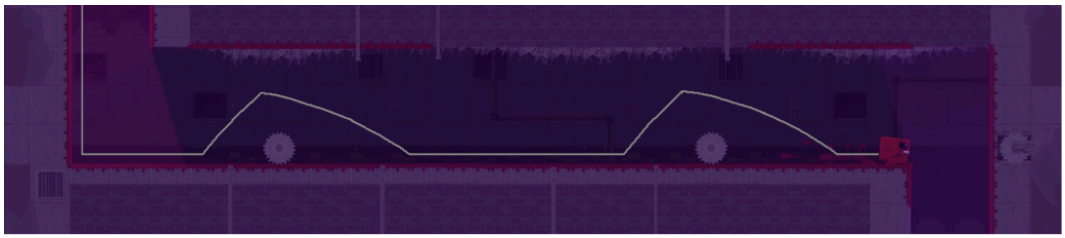


Figure 6: Meat Boy avoiding obstacles.

In general, the tool and the framework are intended to support game designers in iteratively designing a jump that features a desired game feel (Swink 2008). As such they allow highly detailed prototyping by offering an accessible interface that gives immediate feedback to design decisions. The framework and the tool support the exploratory nature of game design and thus contribute to better decision making during the design process. We hope that this line of research leads to more diversity in video games by increasing the variety in movement design and deepening the understanding of existing games.

## BIBLIOGRAPHY


Aldrich, J. 2012a. *A Guide to Super Mario World Physics.* http://s276.photobucket.com/user/jdaster64/media/smw_physics.png.html.

———. 2012b. *Complete Guide To SMB's Physics Engine.* http://s276.photobucket.com/user/jdaster64/media/smb_playerphysics.png.html.

———. 2012c. *LUIGI'S PHYSICS (Super Mario Advance 4)*. http://s276.photobucket.com/user/jdaster64/media/sma4_physics.png.html.

———. 2012d. *Super Mario Bros. 3 Physics.* http://s276.photobucket.com/user/jdaster64/media/smb3_physics.png.html.

Begy, Jason. 2010. "The history and significance of jumping in games." In *Proceedings of the 4th Annual Vienna Games Conference,* 83–96.

Bits & Beasts. 2015. *Feist.* Finji.

Brøderbund. 1989. *Prince of Persia.* Brøderbund.

Butler, Tom. 2014. *The Rise of The Jump, Polygon.* http://www.polygon.com/features/2014/1/20/5227582/the-rise-of-the-jump.

Campbell, Jonathan, Jonathan Tremblay, and Clark Verbrugge. 2015. "Clustering player paths." In *Foundations of Digital Games.*

Capcom. 1988. *Bionic Commando.* Capcom.

D'Angelo, D. 2015. *Shovel Knight: Plague of Shadows Mobility Design.* http://gamasutra.com/blogs/DavidDAngelo/20150831/252543/Shovel_Knight_Plague_of_Shadows_%20Mobility_Design.php%20http://gamasutra.com/blogs/DavidDAngelo/20150831/252543/Shovel_Knight_Plague_of_Shadows_%20Mobility_Design.php.

Daniels, D. 2013. *Game Feel - Part 2.* https://derek-daniels.squarespace.com/blog/2013/2/23/game-feel-part-2.

Delphine Software. 1992. *Flashback.* U.S. Gold.

Eurogamer. 2015. *Miyamoto on World 1-1: How Nintendo made Mario's most iconic level.* https://www.youtube.com/watch?v=zRGRJRUWafY.

Fineberg, D. 2015. *Designing a Jump in Unity.* http://gamasutra.com/blogs/DanielFineberg/20150825/244650/Designing_a_Jump_in_Unity.php.

Hesselgren, A. 2012. *DeadZones: Part One.* http://ludopathic.co.uk/2012/02/28/no-deadzones/.

Isaksen, Aaron, Dan Gopstein, and Andy Nealen. 2015. “Exploring game space using survival analysis.” *Foundations of Digital Games.*

Lefky, A., and A. Gindin. 2007. *Acceleration Due to Gravity: Super Mario Brothers.* http://hypertextbook.com/facts/2007/mariogravity.shtml.

Maxwell, Scott E, and Harold D Delaney. 2004. *Designing experiments and analyzing data: A model comparison perspective.* Vol. 1. Psychology Press.

Monteiro, R. 2012. *The guide to implementing 2D platformers.* http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/.

Moon Studios. 2015. *Ori and the Blind Forest,* Microsoft Studios.

Mossmouth. 2013. *Spelunky.* Mossmouth.

Nintendo. 1981. *Donkey Kong.* Nintendo.

———. 1985. *Super Mario Bros.* Nintendo.

———. 1988. *Super Mario Bros. 3.* Nintendo.

———. 1990. *Super Mario World.* Nintendo.

———. 2003. *Super Mario Advance 4: Super Mario Bros. 3.* Nintendo.

Number None. 2009. *Braid.* Number None.

Olney, M. 2013. *Model of Super Meat Boy’s Physics.* http://mpolney.galineer.com/smb.html#Frame_Step.

Pajot, L., and J. Swirsky. 2012. *Indie Game: The Movie - Online Extra - CONTROL.* https://vimeo.com/34928357.

Pignole, Y. 2013. *The hobbyist coder #1: 2D platformer controller.* http://www.gamasutra.com/blogs/YoannPignole/20131010/202080/The_hobbyist_coder_1_2%20D_platformer_controller.php.

———. 2014. *Platformer controls: how to avoid limpness and rigidity feelings,* http://www.gamasutra.com/blogs/YoannPignole/20140103/207987/Platformer_controls_how_to_av%20oid_limpness_and_rigidity_feelings.php.

Playdead. 2011. *Limbo.* Playdead and Microsoft Studios.

Rogers, T. 2010. *In Praise Of Sticky Friction.* http://kotaku.com/5558166/in-praise-of-sticky-friction.

Saltsman, A. 2010. *Tuning Canabalt, Gamasutra.* http://www.gamasutra.com/blogs/AdamSaltsman/20100929/88155/Tuning_Canabalt.php.

Semi Secret Software. 2009. *Canabalt.* Semi Secret Software.

Sonic Retro. 2014. *Sonic Physics Guide.* http://info.sonicretro.org/Sonic_Physics_Guide.

Sonic Team. 1991. *Sonic the Hedgehog.* Sega.

Stenning, J. 2010. *C# – Screen Capture with Direct3D 9 API Hooks.* http://spazzarama.com/2010/03/29/screen-capture-with-direct3d-api-hooks/.

Stenning, J. 2011. *C# – Screen Capture and Overlays for Direct3D 9, 10 and 11 Using API Hooks.* http://spazzarama.com/2011/03/14/c-screen-capture-and-%20overlays-for-direct3d-9-10-and-11-using-api-hooks/.

Sutphin, J. 2013. *Doing Thumbstick Dead Zones Right.* http://www.third-helix.com/2013/04/12/doing-thumbstick-dead-zones-right.html.

Swink, Steve. 2008. *Game Feel: A Game Designer's Guide to Virtual Sensation.* Amsterdam: Morgan Kaufmann.

Team Meat. 2010. *Super Meat Boy.* Team Meat.

Thompson, T. 2015. *The Legacy of Super Mario Bros.* http://t2thompson.com/2015/06/30/the-legacy-of-super-mario-bros/.

Venturelli, M. 2014a. *Game Feel Tips I: The Ghost Jump.* http://gamasutra.com/blogs/MarkVenturelli/20140810/223001/Game_Feel_Tips_I_The_Ghost_Jump%20.php.

———. 2014b. *Game Feel Tips II: Speed, Gravity, Friction.* http://gamasutra.com/blogs/MarkVenturelli/20140821/223866/Game_Feel_Tips_II_Speed_Gra%20vity_Friction.php.

———. 2014c. *Game Feel Tips III: More On Smooth Movement.* http://gamasutra.com/blogs/MarkVenturelli/20140821/223866/Game_Feel_Tips_II_Speed_Gravity_%20Friction.php.

Warren, J. 2014. *Why Does Mario's Jump Feel So Awesome.* https://www.youtube.com/watch?v=z2oV2DQ2dEA.

West, M. 2013. *Measuring Responsiveness in Video Games.* http://cowboyprogramming.com/2008/05/30/measuring-responsiveness-in-video-games/.

Yacht Club Games. 2014. *Shovel Knight.* Yacht Club Games.