Extracting Physics from Blended Platformer Game Levels

Adam Summerville[1], Anurag Sarkar[2], Sam Snodgrass[3] and Joseph Osborn[4]
[1]California State Polytechnic University
[2]Northeastern University
[3]modl.ai
[4]Pomona College
asummerville@cpp.edu, sarkar.an@northeastern.edu, sam@modl.ai, joseph.osborn@pomona.edu

Abstract

Several recent PCGML methods have focused on generating game levels and content that blend the properties of multiple games. However, these works ignore the fact that blended levels must in some way have blended physics models that enable playable levels. In this work, we present an approach for extracting jump physics models for such blended game domains. We make use of variational autoencoders (VAEs) trained on level data from six platformers, encoded using a previously introduced path and affordance vocabulary. Our results show that the extraction model is able to reasonably recreate the original physics models when given ground truth paths, and is able to produce physics models that can reliably allow an agent to play the generated levels. We also find that there are promising results for blended physics models behaving intuitively between physics models of the original games being blended.

Introduction

While methods for procedural content generation via machine learning (PCGML) (Summerville et al. 2018) were initially motivated by wanting to generate novel content in the style of existing games such as *Super Mario Bros.* (Summerville and Mateas 2016; Guzdial and Riedl 2016a; Snodgrass and Ontañón 2017) and *The Legend of Zelda* (Summerville and Mateas 2015), a new body of work has emerged that focuses on PCGML techniques that seek to leverage trained models to blend existing game domains and/or generate new domains altogether. This has produced works that leverage more creative PCGML approaches such as domain transfer (Snodgrass and Ontanon 2016; Snodgrass 2019), model blending (Guzdial and Riedl 2016b; Sarkar and Cooper 2018), computational creativity (Guzdial and Riedl 2018), training on multiple domains to learn blended domains (Sarkar, Yang, and Cooper 2019) or a combination of the above (Snodgrass and Sarkar 2020).

While some works have included path information, there has been no notion of completing the circle i.e., do the physics latent within generated paths encode a physics model that would allow for playing the level? And if so, how does one extract these latent physics models? Further, while working within the domain of a single game might make this unnecessary e.g., just use the original *Mario* physics when generating *Mario* levels – in blended domains, there is no ground truth physics model to fall back upon. Recently, Sarkar et al. (2020) trained generative models for such blended domains by leveraging a new path and affordance vocabulary that enabled generation of blended levels with paths and jumps. In this work, we directly extend this work by leveraging the jumps found in these generated blended levels to extract physics models for different blended domains. We do this by first generating levels targeting specific games and game blends, with special attention to generating paths that encode the directionality of the path. We then extract physics models that could reasonably have created the generated paths. We test this procedure by comparing the extracted physics to the ground truth physics, and examine the physics of blended domains, seeing how the physics alter with respect to the level geometry.

Related Work

Most prior techniques for procedural content generation via machine learning (PCGML) (Summerville et al. 2018) have focused on learning models for a single game. Such methods have involved using autoencoders (Jain et al. 2016), LSTMs (Summerville and Mateas 2016), GANs (Volz et al. 2018), Bayes Nets (Guzdial and Riedl 2016a), n-grams (Dahlskog, Togelius, and Nelson 2014) and Markov models (Snodgrass and Ontañón 2017) for learning generative models for games such as *Super Mario Bros.*, *The Legend of Zelda* and *Kid Icarus*. In an effort to address generalization and lack of data as well as wanting to discover and create new game domains (similar to e.g. the game blending framework of Gow and Corneli (2015)), more recent works have built models that work with multiple games and domains at the same time. This has included domain transfer (Snodgrass and Ontanon 2016; Snodgrass 2019), game generation (Guzdial and Riedl 2018) and game blending (Sarkar and Cooper 2018; Sarkar, Yang, and Cooper 2019; Snodgrass and Sarkar 2020). Recent work (Sarkar et al. 2020) built on these latter game blending approaches by ex-

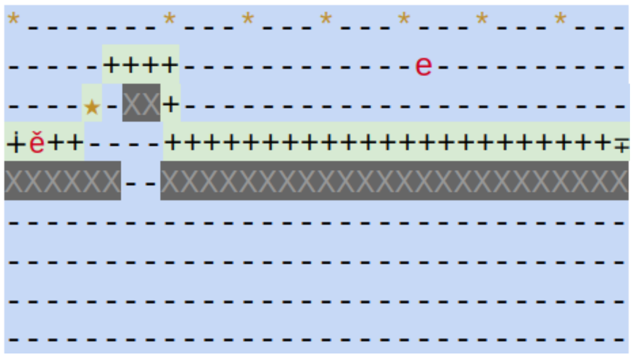


Figure 1: An example of the level representation used (note: color is added only for presentation purposes here). Player path is represented using + with special handling of start (+), end (∓), and places where the path overlaps non-empty tiles ($\check{e}$ for a moving hazard and ⋆ for a collectable).

tending their domain from two to six games, introducing a path and affordance vocabulary and training on levels annotated with A* paths derived from the jump arcs of the respective games. This enabled generation of blended levels spanning all the games while also containing traversable paths and jumps. In this paper, we utilize the paths and jumps in the blended levels generated by this latter approach to extract physics models for the blended domains.

Such physics models have not been the subject of much prior PCGML work with a majority of prior PCGML research focusing on learning models of game levels and only a few attempting to learn models of game physics and game rules. Guzdial, Li, and Riedl (2017) presented an approach termed game engine search for learning the rules of *Super Mario Bros.* using video gameplay data. Summerville, Osborn, and Mateas (2017) learned a hybrid automaton describing the jump physics in Mario. Similarly, Summerville et al. (2017) used data from a Nintendo Entertainment System (NES) emulator to learn automata describing the jump physics of a large number of NES platformer games. To our knowledge, our work is the first to extract such physics models for blended game domains.

Level Data and Representation

For our approach, we used six classic NES platformer games - *Super Mario Bros.*, *Super Mario Bros. II: The Lost Levels*, *Ninja Gaiden*, *Metroid*, *Mega Man*, and *Castlevania* - all represented using the path and affordance vocabulary introduced in (Sarkar et al. 2020), which in turn was derived using the Video Game Level Corpus (Summerville et al. 2016) and the Video Game Affordance Corpus (Bentley and Osborn 2019). Because these games have disparate vocabularies of tiles, we need a common language to describe all of the levels – *solidity*, *climbability*, *passable*, *powerup*, *hazard*, *moving*, *portal*, *collectable*, and *breakable*. These affordances can be combined – e.g., a breakable brick would be "breakable+solid" – which leads to 14 unique combinations (see (Sarkar et al. 2020) for a more detailed description).

A key difference between the level representations found here and in the earlier work of Sarkar et al. (2020) is the representation found here includes not just path information but also the directionality of the path – the starting and ending position found in a segment have a special representation. This allows the downstream physics extraction process to extract the correct physics as paths are not necessarily bi-directional (e.g., very large falls should be represented as such, and not very high jumps). See Figure 1 for an example.

To account for differences in sizes and dimensions of the levels in each game, we used a uniform segment size of 15×32 for all games, adding vertical padding as required. We focused on horizontal sections of levels, thereby ignoring the vertical sections found in *Ninja Gaiden*, *Metroid* and *Mega Man*. After a filtering process to discard duplicate segments and segments mixing discrete rooms, we ended up with 1907 segments for *Mario (SMB)* (referring to both version of Mario mentioned above), 504 segments for *Ninja Gaiden*, 1833 segments for *Metroid*, 924 segments for *Mega Man* and 775 segments for *Castlevania*.

Generative Model

For generating levels from which to extract physics, we used a Gated Recurrent Unit-Variational Autoencoder (GRU-VAE), implemented using PyTorch (Paszke et al. 2017). The encoder consisted of 3 hidden layers of size 1024 while the decoder had 2 hidden layers of size 256—both using a dropout rate of 50%. To help with convergence, the variational loss was annealed linearly from 0 to 0.05 times the variational loss over the first 5 epochs before the rest of the training continued at that rate—for a total of 50 epochs using the Adam optimizer and a learning rate of 10^{-5}. At decoding time, the decoder is initialized with a latent embedding and then decodes in an auto-regressive manner with sampling. For each generation, we sampled 10 segments and kept the one with the lowest perplexity (highest likelihood).

Physics Extraction

To extract the physics, we must first define the "physics" of a static level. In part, this seems ridiculous, as a static level cannot have a conventional physics model, as there is no notion of time. However, while this seems like an intractable problem, we believe that for several platformer games, there is an implicit correlation between horizontal position and time – e.g., a speedrunner of Mario is almost always moving to the right as quickly as they possibly can. In fact, the A* agent that we use to simulate "playing" the levels also operates under this assumption. Thus, we think it is reasonable to relax the physics models from a notion of y position versus time to a relation of y position to x position – with the understanding that the x position is supposed to be constantly progressing in the direction of the goal. It is important to note that the "physics" model we are extracting actually supports an infinite number of different possible physics models – changing the maximal x speed will result in different physics models. Some games have much slower horizontal speeds (*Castlevania* has a maximal horizontal speed of $\sim$3.7 tiles per second), while others have much faster speeds (*Super Mario Bros.* has a maximal horizontal speed of 10 tiles

Standard Physics Model	**Extracted Physics Model**
Parameters	
Impulse ($\frac{\partial y}{\partial t}$)	Impulse ($\frac{\partial y}{\partial x}$)
Gravity ($\frac{\partial y}{\partial t^2}$)	Gravity ($\frac{\partial y}{\partial x^2}$)
Assumptions	
Player has control over height of jump	Player takes the highest possible jump
Player can alter horizontal position during jump	Player is always moving at maximum horizontal speed

Table 1: A comparison between the *standard* physics models as found in platformer games and the extracted physics models produced here.

per second) – the rest of the games we looked at have speeds of around 5.5 tiles per second. If one wished to take these extracted physics and use them in a playable game, the different x speeds would result in different feeling games, but somewhere in the 4 to 10 tiles per second range would result in games playable by humans.

We also note that a large number of platformer games allow for the player to control the arc of the jump based on how long they hold the jump button – in this work, *Super Mario Bros.*, *Metroid*, and *Mega Man* all allow for this, while *Castlevania* and *Ninja Gaiden* do not – and this notion of player control is not contained within the static maps. Again, we make the simplifying assumption that higher jumps are preferred – we want to determine the frontier of what space is reachable. Table 1 describes the "physics" in contrast to the standard physics found in the game.

One final note – many games have different physics models when the player is falling in their jump versus when they are in the rising portion of their jump – e.g., in *Super Mario Bros.* gravity can more than double when the player is falling. As such, we learn a separate gravity value for the rising and falling portions of a jump.

Extraction

Having defined the physics model of a static level, we now discuss the process for extracting said physics model. To determine how the path represents the player's position through time, a Breadth-First Search is performed, beginning from the start position and progressing until the end position is found. This provides a coarse notion of the progression of the path. The path is then followed and the algorithm described in Figure 2 is used to separate the portions of the path that are (1) *grounded*, (2) *jumping*, and (3) *falling*.

Once segmented, the segments are filtered to remove noisy jumps:

- Any jump or fall of two or fewer data points is removed – these are too small to derive any useful physics from
- Any segment that moves more than 2 tiles in a single step is removed – these represent "broken" paths and are likely to represent a corruption of the physics

For each x position in a jump, the highest corresponding y value is recorded – e.g. if a jump consists of $[[0,0],[0,1],[1,1],[1,2]]$ then the highest recorded positions per x value are $[[0,1],[1,2]]$. This is done for all jumps and the statistics for the x positions found across all jumps are calculated. Jumps are then scored by how many of their y positions agree with the $P-$percentile y values across all jumps. P is then a hyperparameter that can be tuned to determine what one expects to see from the jumps – given that 3 of the games have variable height jumps, our inductive bias is that jumps higher than the median should be selected given that we wish to find the upper extents of possible jumps. We filter jumps that have more than 50% disagreement with the $P-$percentile jump. In the next section, we discuss the criterion for the selection of P. Finally, given the filtered jumps and falls, we perform an Ordinary Least Squares regression where the dependent variable is y position and the independent variables are x (corresponding to Impulse) and x^2 (corresponding to Gravity).

```
function SEGMENT EXTRACTION(path)
                                        → $jumps, falls$
    $l$ ← path[0]
    $g_p$ ← onGround($l$)
    $y_p$ ← $l.y$
    $jumps$ ← []
    $jumping$ ← not $g_p$
    $jumping$ ← not $g_p$
    $falls$ ← []
    $seg$ ← [$l$]
    for $l$ in path[1:] do
        $g$ ← onGround($l$)
        $y$ ← $l.y$
        $seg$.append($l$)
        if $g$ and not $g_p$ then                ▷ Landed
            $jumping$ ← False
            $falling$ ← False
            $falls$.append($seg$)
            $seg$ ← [$l$]
        else if not $g$ and $y > y_p$ then        ▷ In Jump
            if not $jumping$ then
                $jumping$ ← True
                $falling$ ← False
                $seg$ ← [$seg[-1]$]
            end if
        else if not $g$ and $y < y_p$ then        ▷ Falling
            if $jumping$ then
                $jumps$.append($seg$)
            end if
            if not $falling$ then
                $falling$ ← True
                $jumping$ ← False
                $seg$ ← [$seg[-1]$]
            end if
        end if
    end for
end function
```

Figure 2: Pseudocode describing the segmentation of jumping, falling, and being on the ground from path positions

Game	Original	Generated
Castlevania	1.03	4.98
Super Mario Bros	0.71	0.25
Metroid	0.93	0.94
Mega Man	0.22	2.94
Ninja Gaiden	0.80	1.03
Total	3.69	10.14

Table 2: Root Mean Squared Error (RMSE) for y values per x value for the physics models extracted for the original levels and the generated levels when compared to the actual game physics. We also see Castlevania is difficult to extract, in part because its arc does not actually follow a parabola.

Evaluation/Discussion

To evaluate the extracted physics, there are a number of concerns.

1. **Faithfulness of the Generated Physics** – Do the paths contained in segments generated targeting a specific game domain faithfully recreate the physics found in the original segments?
2. **Validity of the Extraction Process** – Is the extraction process capable of reconstructing the original jump parameters from the training data (where the paths were generated from an agent using the original parameters)?
3. **Interpretability of Blended Physics** – Do the segments found in interpolations between the original games result in physics that are interpolated between the games?

Faithfulness to Original Physics

To assess whether the original physics can be extracted, we first use the extraction process on the training segments – these should have the physics flawlessly encoded within them. We ran a hyperparameter grid search over the P-percentile to ascertain what percentile leads to the most accurate physics model. We assess the accuracy of the physics model by calculating the Sum of Squared Error (SSE) for y values per x value for the jump models produced by the physics models extracted for the original segments. $P = 75\%$ led to the lowest total SSE summed over all of the games, although different games had differing values (From *Castlevania* at 60% to *Super Mario Bros.* at 80%). However, the mean value of the percentiles was 72.6%, so we feel comfortable with using the 75th percentile jumps for the physics model (which confirms our inductive bias that we wanted jumps higher than the median).

Of course, in some sense, the aesthetics of the reconstructed jump arcs are more important than the error – most importantly, do the extracted jumps result in the same maneuvering and reachability as the true physics? Figure 3 shows the true jump arcs in comparison with the jump arcs extracted from the original segments. We see that the extracted jump for *Metroid* (orange) reaches the same heights, but does not reach quite as far horizontally, due to a higher falling gravity. We see that the extracted jump for *Mario* (red) is a bit short in height (reaching only $\sim$3.5 tiles high instead of 4 tiles in height) but has the same horizontal space covered. *Mega Man* (light blue) has a slightly different arc, but reaches the exact same height and has the same horizontal space. *Ninja Gaiden*'s (dark blue) extracted jump falls short of the true jump both in height (reaching a maximum of 3.8 tiles instead of 4) and in distance (9 tiles instead of 10). Finally, *Castlevania*'s extracted jump has the same horizontal reach, but reaches higher (2.8 tiles instead of 2 tiles).

Generally, these jumps would support much of the same gameplay, although the height differences for *Super Mario Bros.* and *Ninja Gaiden* would need to be bumped up to the nearest whole number of tiles to have the same gameplay. With this, we feel satisfied that the physics extraction process works well enough to faithfully extract physics that would support playing the game, and we turn our attention to the generated levels, to see how faithfully they are able to represent the physics.

Latent Reconstructions

To evaluate the generated physics, we sampled the generative model to produce level segments that were from the latent space of the encoding corresponding to each game. To do this, we first obtained the latent encoding for every level segment from a given game. We then calculated the mean and standard deviation for these encodings. Finally, we sampled 2000 encodings from a normal distribution with the calculated parameters – these encodings were then decoded into level segments. As a note, the level segments were sometimes lacking in a beginning and ending (due to the stochastic nature of the generation process) – these segments were excluded from the physics extraction process as it is impossible to determine the progression of the generated path – in all, this led to the dropping of 326 segments in total (3.26% of the generated segments) with *Metroid* having the highest proportion of corrupted segments (8.1%). The physics models were extracted using the same 75th percentile criterion as computed in the original levels (no hyperparameter search). Table 2 shows the RMSEs between the physics models extracted from the original and generated segments when compared with the actual game physics. Both errors are broadly comparable (and is in fact lower for generated segments in the case of *Super Mario Bros.*).

As noted above, the aesthetics of the reconstructed jump arcs are as, if not more, important than the errors. Figure 4 shows the true jump arcs in comparison with the jump arcs extracted from the generated segments. We see that the extracted jump for *Metroid* (orange) reaches the same height, but does not reach quite as far horizontally, due to a higher fall gravity. We also see that the extracted jump for *Mario* (red) reaches the same height but extends horizontally.

We note that for the rest of the games, the generated arcs show a regression towards the physics of *Super Mario Bros.*. The generated *Mega Man* and *Castlevania* arcs are nearly identical to the *Super Mario Bros.* arc. Finally, we see that the generated *Ninja Gaiden* (dark blue) arc is very similar to the one extracted from the original segments reaching not quite as high but having the same horizontal duration.

Again, generally, these physics would support the same gameplay – in fact *Super Mario Bros.* would be playable as is with no intervention with the model extracted from the

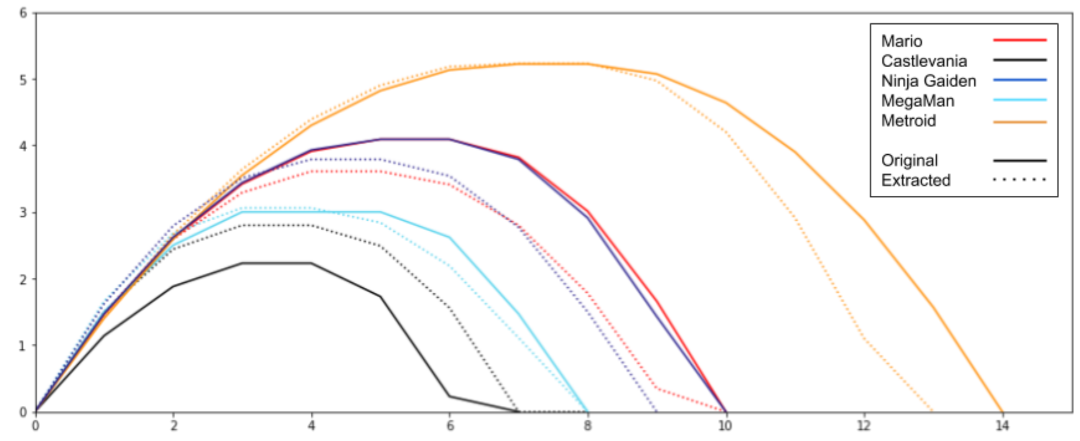


Figure 3: The jump arcs for the true physics (solid lines) compared to the extracted training jumps (dotted lines).

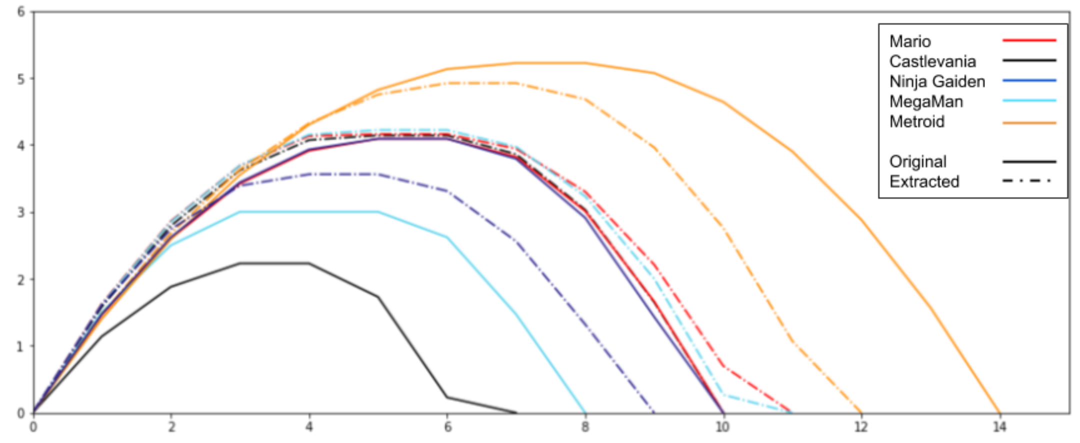


Figure 4: The jump arcs for the true physics (solid lines) vs the extracted generated jumps (dash-dotted lines).

generated levels (unlike the model extracted from the original segments). Also, while the models for *Castlevania* and *Mega Man* are more lenient for the generated extractions than the true physics, the levels would be playable with the extracted models.

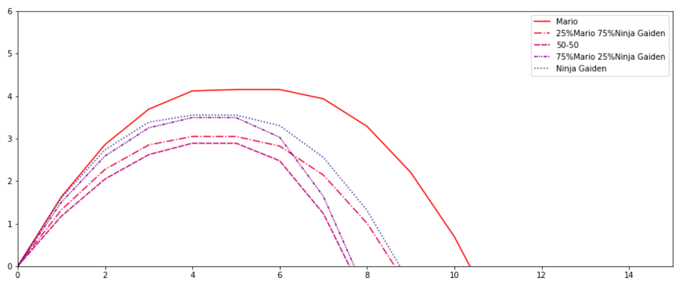
Blended Physics

Unlike the above categories, there is no direct comparison to see how well the extracted physics recreate the original physics – instead visual inspection is the best way to assess the interpolated physics. Figure 5 shows the physics extracted from interpolations between different games. To get the interpolations, we take 10 segments from each game, encode them into the latent space, interpolate between all pairs of encodings at 25%-75%, 50%-50%, 75%-25%, and then decode 20 times (since the decoding process is stochastic, the same encoding can produce different segments). We note that for most pairs, the interpolated physics seems to settle into a jump that is actually unlike the exemplars, but relatively stable across the blends – a jump that reaches about 3 tiles in height and 8 tiles in width (which is actually quite similar to the jump of *Mega Man*). This jump is somewhat average across the games (although a jump of 3.5 in height and 9 in width would be closer to average), so it seems that most blends actually go through a sort of in-between average space that just encodes generic platformerness as opposed to any real per-game-pair specific physics. That being said, *Metroid* – being the most extreme of the physics – does tend to have some blends that incorporate its higher and longer jumps – namely, blends with *Castlevania* (Figure 5h), *Mega Man* (Figure 5j) and *Ninja Gaiden* (Figure 5b).

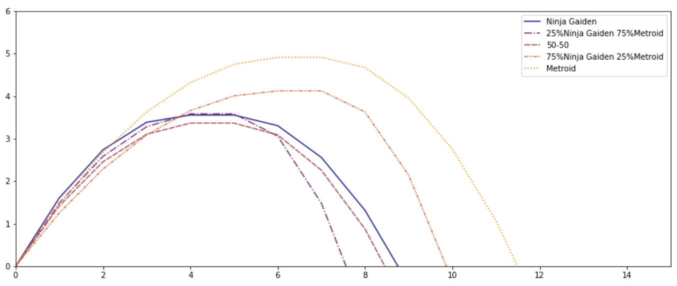
Conclusion and Future Work

In this paper, we presented a method for extracting “physics” from static levels of the kind often used in PCGML level generation. We compared the extractions from both ground truth training examples and generations to the ground truth physics. In addition, we explored the physics found within blended domains, with some promising examples of blended physics.

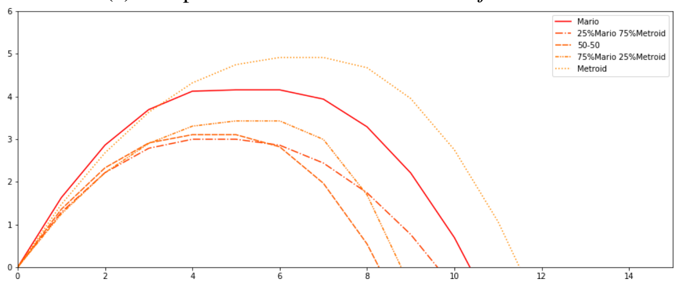
In the future, we would like to expand this work to explore different level orientations (e.g., vertical levels found in *Kid Icarus*). We would also like to explore the inverse process – given a physics model, generate levels that are playable.



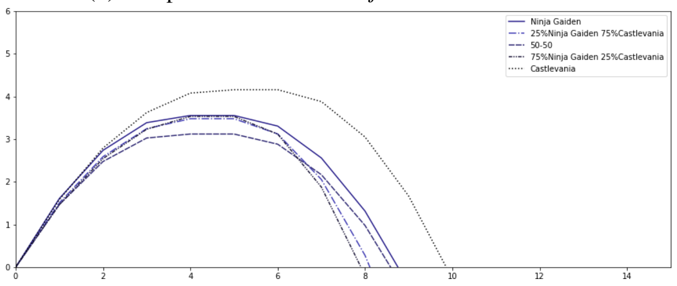
(a) Interpolation between *SMB* and *Ninja Gaiden*



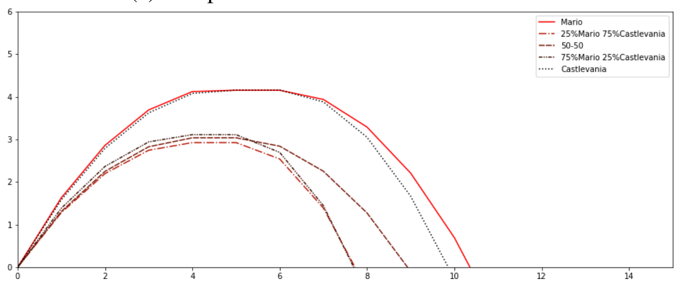
(b) Interpolation between *Ninja Gaiden* and *Metroid*



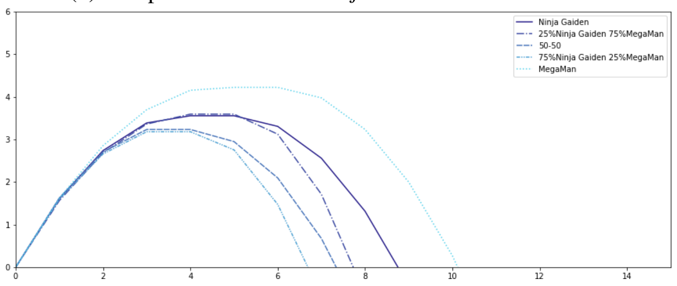
(c) Interpolation between SMB and Metroid



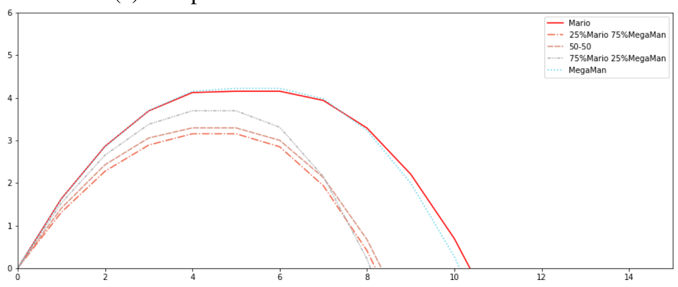
(d) Interpolation between *Ninja Gaiden* and *Castlevania*



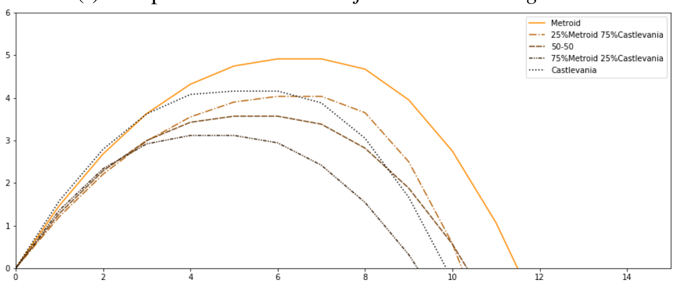
(e) Interpolation between *SMB* and *Castlevania*



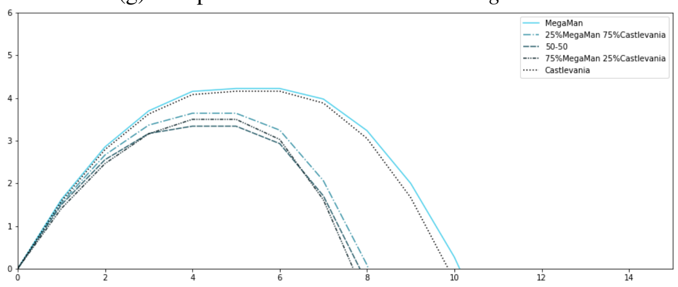
(f) Interpolation between *Ninja Gaiden* and *Mega Man*



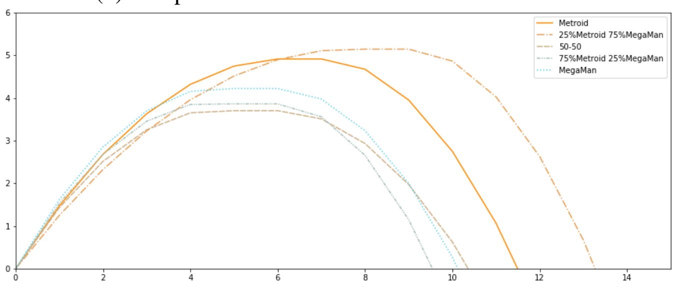
(g) Interpolation between *SMB* and *Mega Man*



(h) Interpolation between *Metroid* and *Castlevania*



(i) Interpolation between *Mega Man* and *Castlevania*



(j) Interpolation between *Metroid* and *Mega Man*

Figure 5: The jump arcs for the interpolations between games. We note that many of the blended jumps form a sort of "average" jump (found in (a), (b), (c), (d), (e),(f), (g), and (i)) where the jump reaches about 3 tiles in height and 8 tiles in length. However, some blends have more interesting, intuitive blends, such as those between *Metroid* and both *Castlevania* and *Mega Man*.

References

Bentley, G. R., and Osborn, J. C. 2019. The videogame affordances corpus. *2019 Experimental AI in Games Workshop (EXAG).*

Dahlskog, S.; Togelius, J.; and Nelson, M. J. 2014. Linear levels through n-grams. *Proceedings of the 18th International Academic MindTrek.*

Gow, J., and Corneli, J. 2015. Towards generating novel games using conceptual blending. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).*

Guzdial, M., and Riedl, M. 2016a. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).*

Guzdial, M., and Riedl, M. 2016b. Learning to blend computer game levels. In *Seventh International Conference on Computational Creativity (ICCC).*

Guzdial, M., and Riedl, M. 2018. Automated game design via conceptual expansion. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).*

Guzdial, M.; Li, B.; and Riedl, M. O. 2017. Game engine learning from video. In *26th International Joint Conference on Artificial Intelligence (IJCAI)*, 3707–3713.

Jain, R.; Isaksen, A.; Holmgard, C.; and Togelius, J. 2016. Autoencoders for level generation, repair and recognition. In *ICCC Workshop on Computational Creativity and Games.*

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *Conference on Neural Information Processing Systems (NeurIPS) Autodiff Workshop.*

Sarkar, A., and Cooper, S. 2018. Blending levels from different games using LSTMs. In *2018 Experimental AI in Games Workshop (EXAG).*

Sarkar, A.; Summerville, A.; Snodgrass, S.; Bentley, G.; and Osborn, J. 2020. Exploring level blending across platformers via paths and affordances. In *Sixteenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).*

Sarkar, A.; Yang, Z.; and Cooper, S. 2019. Controllable level blending between games using variational autoencoders. In *2019 Experimental AI in Games Workshop (EXAG).*

Snodgrass, S., and Ontanon, S. 2016. An approach to domain transfer in procedural content generation of two-dimensional videogame levels. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE).*

Snodgrass, S., and Ontañón, S. 2017. Learning to generate video game maps using Markov models. *IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG).*

Snodgrass, S., and Sarkar, A. 2020. Multi-domain level generation and blending with sketches via example-driven BSP and variational autoencoders. In *Fifteenth International Conference on the Foundations of Digital Games (FDG).*

Snodgrass, S. 2019. Levels from sketches with example-driven binary space partition. In *Fifteenth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE).*

Summerville, A., and Mateas, M. 2015. Sampling Hyrule: Sampling probabilistic machine learning for level generation. *Tenth International Conference on the Foundations of Digital Games (FDG).*

Summerville, A., and Mateas, M. 2016. Super Mario as a string: Platformer level generation via LSTMs. *Proceedings of 1st International Joint Conference of DiGRA and FDG.*

Summerville, A. J.; Snodgrass, S.; Mateas, M.; and Ontañón, S. 2016. The VGLC: The video game level corpus. In *Seventh Workshop on Procedural Content Generation at First Joint International Conference of DiGRA and FDG.*

Summerville, A.; Osborn, J.; Holmgård, C.; and Zhang, D. W. 2017. Mechanics automatically recognized via interactive observation: Jumping. In *Twelfth International Conference on the Foundations of Digital Games (FDG)*, 1–10.

Summerville, A.; Snodgrass, S.; Guzdial, M.; Holmgård, C.; Hoover, A. K.; Isaksen, A.; Nealen, A.; and Togelius, J. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games (ToG).*

Summerville, A.; Osborn, J.; and Mateas, M. 2017. Charda: Causal hybrid automata recovery via dynamic analysis. In *26th International Joint Conference on Artificial Intelligence (IJCAI).*

Volz, V.; Schrum, J.; Liu, J.; Lucas, S. M.; Smith, A.; and Risi, S. 2018. Evolving Mario levels in the latent space of a deep convolutional generative adversarial network. In *Genetic and Evolutionary Computation Conference (GECCO)*, 221–228. ACM.