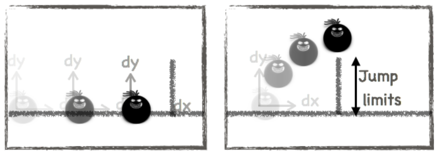
Game design as CPS

Vidya Narayanan
vidyan@cmu.edu

May 9, 2017

Abstract

2D Platform and side scroller games such as Super Mario or Flappy Bird are very popular. Designing such games is an elaborate and creative process but a few simplified steps of interest include – the design of the primary character which usually includes simplified physics, the design of the level or map, the rules of the game and its challenges.The goal of this project was to explore if some aspects of game building can be verified using hybrid systems and games. This project attempts to develop a simple side scrolling game with a single character that the player can control. I modeled a way to generate simple but *arbitrary* levels and showed that they are *safe*. I also modeled two actions for the character that shows that the player can achieve game challenges that require the character to avoid (by jumping over) or attack(by jumping onto) a specified target. This project provided an opportunity to further understand the diamond modality and adversarial components when modeling. While each of these actions are proven with a fixed number of steps, composing these features would provide a starting point to design a simple game such that its non-triviality and playability can be proven by the designer.



1 Introduction

Motivation Games appear to be non-critical when compared to traditional domains that involve cps such as automobiles and robotics, but it offers an excellent framework to understand many aspects of cps. Many games are built assuming simplified physics but have a consistent model or view of the world. Even though there are hacks thrown in, these are generally incorporated as a part of the model – for example, a certain region of the game may have a different value for gravity but the physics of the game continues to model acceleration due to gravity [6]. The game physics lends itself to a continuous system that can be modeled using ordinary differential equations. The human or computer player interacts with game elements through a game controller. There is certainly latency and lag that has to be considered, making the controller inherently time triggered even though sensing in a virtual world can be assumed to be perfect. Thus, hybrid systems provides a suitable framework in which one could think about verification of these games.

Design Hybrid games involve understanding constraints and strategies under which a player could ensure her win. These games include could also include an adversarial element that the character is playing against. While designing a game, the game designer and the choices she incorporates into the game are adversarial to the player. This makes design inherently a hybrid game rather than a system even if the game itself does not involve an explicit adversarial character.

Playability Winning a game requires that the human player controlling the game performs conscious actions towards achieving the goal of the game. Naturally, this provides an opportunity to understand the *diamond* modality further. We refer to this liveness property as *playability* since it ensures that the player is capable of playing this game.

Safety The camera movement in side scrolling platform games tend to follow the central character of the game without much movement along the orthogonal axis. This is also important to have a pleasant viewing experience and fair game play where the character is constantly visible but the camera does not rapidly move too much. Hence, it was natural to view the notion of *safety* as ensuring that the game is designed such that the characters activity can be constrained in the vertical space while continuing to be playable – i.e perform actions crucial to getting the character around. One could also consider a case where a similar game is set in the physical world – such as in a play gym – notions of safety becomes more relevant in such a case.

Education Physics based games have been used as an amusing medium for physics and computer science education in general [5]. I believe they also provide an excellent learning ground for understanding cyber-physical systems and differential dynamic logic. Students can build upon their understanding of simple-event driven hybrid systems, safety properties, time-triggered controllers, liveness properties, adversarial elements and winning strategies within the a unified context of a game. I would hope that the allure of having a verified game design and controller at the end of the term would make learning even more interesting.

2 Related work

Researchers across various fields – artifical intelligence, complexity and game theory, graphics, have shown interest in building and proving games. Aloupis and colleagues [2] have shown that classic nintendo games such as *Mario* and *Donkey Kong* are NP-hard. Particularly, they discuss the problem of *reachability* – given a stage or a level, can the player reach point t from point s. Given that this is hard, discussing if the level is *playable* may seem doomed to begin with. The goal of this project is orthogonal to this notion of reachability, it is to help design a level ensuring a simple notion of reachability in a bottom-up compositional manner. Moreover, for an explicitly encoded, constant screen size version of these games, Demaine and colleagues show that these games are solvable in polynomial time.

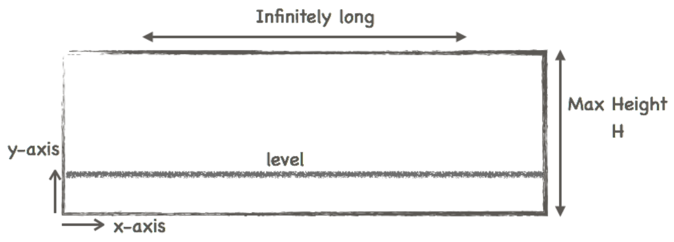
Adelhart [1] formally verifies reachability from the start to the end of a map by construction. That is, they show that the levels generated by their system is *correct* even though an arbitrary level may be difficult to verify. This is similar to our goal and though not in the context of hybrid games, they verify game construction by analyzing a graph over actions where each action can be verified by a simple axiom. This is much like the approach we would like to take here, the individual actions are proven as smaller hybrid games. The composition of these smaller games into a full game is out of the scope of this project but a similar graph based approach over smaller hybrid games can be considered in the future. Muller [7] and colleagues introduce techniquess to reason about component based modeling and verification of hybrid systems where components have local responsibilities. Such an approach might be crucial to verifying large systems like games that can quickly get out of hand when adding design complexity.

Hybrid games and systems have been successfully used to model and verify airplane dynamics and collision handling systems, robotic surgeries etc. While seemingly unrelated to the domain of game design, primary notions of reachability and actions such as avoiding or targeting a spatial region is common to these physical systems. The piecewise linear modeling of safe-flying trajectories can be used to generate game maps and player motion [4]. In this project, we allow a highly general but simple notion of a game level consistently constrained to ensure safety and playability.

3 Design

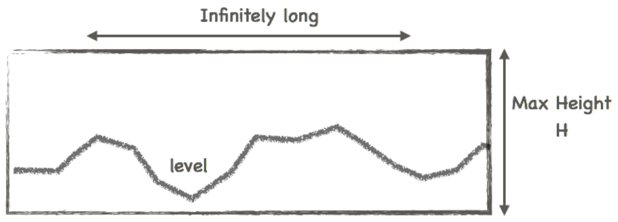
We first informally describe the game model that we consider.

Extent The game *extent* is defined as bounding height that can be fixed based based on the screen resolution one wants to consider. The game *scrolls* along the $x-$axis and is considered to be infinitely long as shown below:

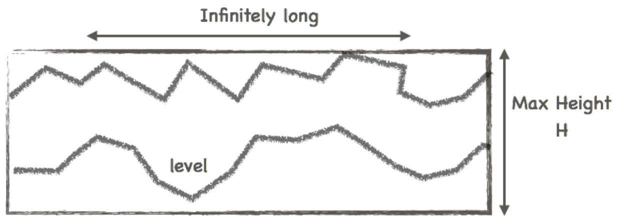


An arbitrary point along the axis can be chosen to be a final goal state or time limit is fixed to make the game play finite. The game *world*, and *level* are used interchangeably to represent a side view of the ground over which the player can *walk*. The walking motion is one-dimensional. The schematic above shows a simple line, the game can allow the following styles of level design:

Level design First, the slope of the level can be changed to allow randomly generated levels. We refer to these as *sloped-level*. An example of such a level is shown below:

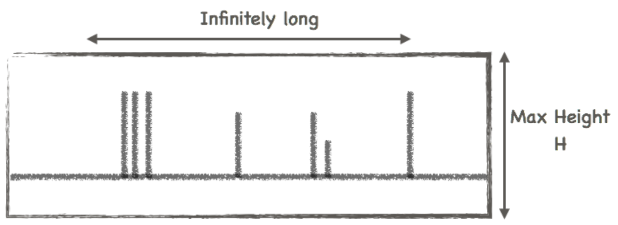


Further, within the fixed height span H, the design can also incorporate ceiling restrictions by generating an additional ceiling slope as follows:

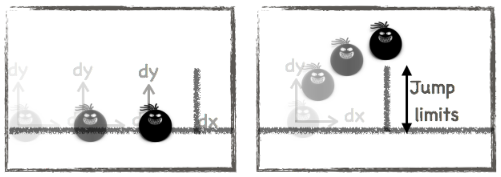


We refer to these as *doubly-sloped level*.

Additionally arbitrary obstacles can be added to the level ground allowing procedural generation of levels such as :



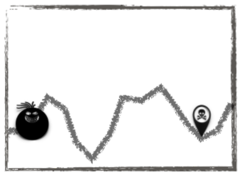
Player The player is represented in the model as a point mass with two coordinates representing its position along the x-axis and y-axis. A unit vector represents the direction of its motion. The character can move with a constant minimum positive velocity, alternately the velocity can be changed by using the game controller. The maximum speed of the character is capped by a game defined constant. A radius to define the spatial extent of the character may be considered, it is avoided in the model for simplicity. I use the term character and player interchangeably. The character can also be made to jump, in which case, a velocity is added along the y-axis making the character jump. When the character is in the air, gravity is turned on, forcing it to land back on the ground.



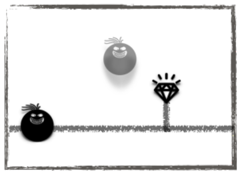
An important consideration when designing the levels is that any obstacle placed must be reachable given the characters linear and jump velocity bounds. Level design above does introduce challenges such as obstacles that may have to be avoided or reached as a part of the game play. In order to accomplish these goals the players game controller allows the player to *walk*(along the ground or level) and *jump*(across and over) obstacles. These are the allowed player *actions.*

Challenges The obstacle based level indicated challenges that would require the player to jump. We now explicitly define the two *challenges* the player faces in this game.

The game design can mark regions or points of the level as dangerous that the player needs to avoid. Given the actions allowed, the only choice is to jump over these regions. We refer to these as *avoid*-challenges. For example, the level design can require that points marked with the danger sign need to be avoided by the player in order to survive the level.



The second challenge requires the character to land on a target (obstacle, platform, jewels or enemies) in order to succeed. Again, if the target is not at the level ground, the player has to jump in order to achieve this goal. We refer to these as *attack* challenges in general. Here, the player's goal is to target the jewel.



The *game* consists of a level, player and a set of challenges. We next show how these are modeled in differential dynamic logic using the KeyMaeraX proof system. We then discuss proofs for level safety given player definition and for individual challenges.

4 dL models and proofs

Player model As mentioned earlier, we model the character with two coordinates (x, h) representing its position along the x-axis and its height. The player is equipped with a unit vector (dx, dy) that represents the direction of motion i.e the slope along the level. The player has a velocity v. The game is forward scrolling, hence the player can stop or move forward which bounds the velocity by 0 from below. The player moves forward following a simple ODE

$$\boxed{x' = v \cdot dx, h' = v \cdot dy}$$

The game design also bounds the velocity by same constant V from above. The game controller allows the player to jump. This is incorporated by adding a jump velocity j that takes a game defined value to the h component of the motion. Thus, the player moves and jumps based on the ODE.

$$\boxed{x' = v \cdot dx, h' = j + v \cdot dy}$$

When the character is walking on the ground gravity does not impact motion since we wan't the player to be on the ground. When the player choses to jump, his jump velocity j is affected by acceleration due to gravity g.

$\boxed{j' = -g}$

We enable gravity only when the player starts to jump and switch it off once the player lands on the ground. Further, we only allow the player to jump when on or close to the ground(indicated by l). The game controller is thus defined as

```
/* player controller */
{
        /* when on the ground, jump or stay on the ground */
        if( h= l ){ {j:=J; g:=G;} ++ {j:=0;g:=0}}
        /* increase or decrease velocity*/
        { {v:=v+1;} ++ {v:=v-1;} ++ {v:=v;}}
        /* clamp velocity to bounds*/
        if(v>V){v:=V;}
        if(v<0){v:=0;}
}
```

Finally, once the player starts his jump, gravity is turned on by setting the value of g to the gravitational acceleration constant defined within the game. Once the player lands back on the ground l, this needs to be turned off so that the player can continue on the level. Since this is a part of the worlds reaction on impact, we handle this by letting the world *fix-up* the constants. We add a domain constraint to the ODEs

$\boxed{...\&h >= l}$

that triggers a world fix-up as

```
/* world fix-up controller */
{
if(h<l){ h:=l; g:=0; j:=0;}
}
```

Level model We model the level design as being procedurally generated as a hypothetical player moves forward. Hence the level is defined by its slope (dx, dy) at the player position. The ceiling is also generated as this player moves forward with its own slope defined by (udx, udy). The level height is defined by a variable l that mimics the players motion along the ground and is maintained explicitly to ensure the level can propogate correctly even when the player choses to jump. The horizontal position of the level while it is being designed can be set to the players position. Hence, we include a term for level propogation in the ODEs

$\boxed{l' = v \cdot dy}$

The level is a part of the world or the environment and can freely chose to modify its slope. This is done by non-deterministically assigning the lower and upper unit slopes.

```
/* world controller */
{
/* assign an arbitrary slope */
dx:=*; dy:=*;
udx:=*; udy:=*;
/* ensure these are unit vectors*/
?(dx^2+dy^2=1);
?(udx^2+udy^2=1);
/* ensure movement is strictly forward */
?(dx>eps); ?(udx>eps);
}
```

Further, the world must be allowed to modify its slope at will, but the player's controller can only be triggered after fixed units of time. Thus, our environment and world are event driven while the player is time triggered defined by a constant T. To allow for a time-triggered controller we maintain a clock variable t that evolves, along with the other ODEs, according to

$\boxed{t' = 1, ...\&t <= T...}$

To ensure that the player is time-triggered and the world can be event-triggered, we set up the controllers and dynamics as follows:

```
{
        if(t>=T){
                t:= 0;
                /* reset clock and run player controller */
        }
        {ODEs & h>=l & t<=T}
        {/*run world fix-up controller */}
        {/*run world controller*/}

}
```

One can also consider a global clock that is initialized at $clock = 0$ and is evolved as $clock' = 1$. This variable can be used to model time constrained goals that requires the player to be *efficient*. I have not considered efficiency as a part of this game model.

Safety Now that we have defined the player and the level, we can talk about defining safety, what are the necessary constraints to generate safe and playable levels. For simplicity, let us consider a single slope-level.

As suggested earlier, one of the requirements of the level design is to fit within the space confines $0 - H$ along the y-axis. Hence a safe level is one where the ground l is such that $l >= 0 \& l <= H$ and the player h is such that $h >= 0 \& h <= H$. However this means that the player can only jump when there is sufficient distance between l and H. Further more, since level design is arbitrary, this could lead to levels where the player can only move since the level leaves no space for jumping within the confines of the world.

In order to allow the player free choice of movement action, level design should encode the space required to jump while generating levels.

When the player choses to jump, an initial velocity $j = J$ is added to the y-component that then evolves according to $j' = -g$ with $g = G$ until the player hits the ground again. The vertical distance covered by the player while jumping is defined by. We consider the maximum velocity as $J + v$ since we wan't to allow the player to move forward and dy is bounded by 1. For the safety of the level we only want to allow the player to move forward not necessarily by the maximum speed, so we set v to 1.

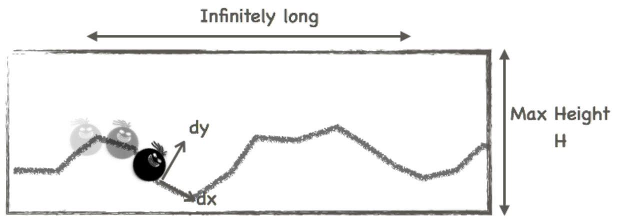
$$\boxed{H_j = (J + v)^2/(2G)}$$

Thus, if $h + H_j <= H \& h >= 0$, the level generated is playable and safe.

$$\boxed{\textbf{safety: } h + H_j <= H \& h >= 0}$$

Note that h is determined by l because of the domain constraints and identical motion ODEs. l is determined by the slope (dx, dy). First, we set the following constraint on the slope generation that for a unit time, the level is safe under moving forward i.e $v > 0$. We then only allow slopes that when followed for T units of time under velocity v, does not exceed the bounds.

$$\boxed{\begin{array}{l} h + hj + v \cdot dy <= H \\ h + v \cdot dy >= 0 \end{array}}$$



Note that considering a positive velocity is important since it ensures that a level is not simply safe because the player can stand in place and jump, the player can also move forward at some velocity. While it might appear as if the level is only safe when the player is moving at velocity $v = 1$, the intent is to generate levels that are *playable* i.e some forward and perpendicular motion is allowed. One can relax this constraint by using the velocity bound V instead of 1 that allow more conservative levels. Putting it all together, the dL can be formulated as

```
(x = 0 & h = 1 & constants) ->
[
{
{/* slope controller */}
{ODEs & t<=T}
}*
] (safety)
```

This proof follows directly by construction and proves in KeyMaeraX using the *solve* rule and QE (.kyx and .kya files have been submitted). Doubly sloped levels prove similarly however require that the position of the ceiling be considered instead of H.

The levels generated so far generate no obstacles. The motion of the player along the level does not require the player to jump though we have proven that the player could jump safely. In order to randomly generate obstacles, we need to consider how far spaced the obstacles need to be such that the player can jump over them. This is important since we only allow the player to jump when she is on the level. However, as long as obstacles are not higher than the extents H, this does not affect the *safety* of the game which is defined on the basis of the players position and does not consider if the player could reach that position at all or not. So such obstacle based levels are safe as long as the obstacle height is within the specified bounds. We next consider the more challenging aspect, can the player succeed challenges assuming a safe level.

Challenge Safety only shows that these components are not placed such that jumping over them would require the character to move off-screen or that it is utterly impossible to jump over them given the maximum height the character can jump to. It is an entirely different matter to show that the player can actually achieve these goals.

Before we proceed towards this goal, we revisit the game model in the context of diamond modalities. Consider a proof for a simplified challenge that requires the player to move forward. One could describe this model schematically as:

```
(initial game state & constraints) ->
<{
{control player}
{ODEs & h>=l & t<=T}
{control world}
}*
> (challenge succeeds)
```

But, this would imply that the player gets to chose all world control decisions in order to achieve success. In some sense, this is not completely inaccurate. It models the following question – Does there exist some random level that the game could generate that the player could win with some choice of actions ? However, when looking at designing games one is mostly not asking that question but rather – Given any level that the game could generate, can the player succeed with some choice of actions ? This would ensure that every level is playable in some sense, even if it required an extremely skilled player. The key point is that even for a simple action of *can the player move forward ?* one needs to introduce an adversarial world that could throw *any* game level at the player.

A better abstraction to encode this is to consider the game model as follows:

```
(initial game state & constraints) ->
<{
{control player}
{ODEs & h>=l & t<=T}^@
{control world}^@
}*
> (challenge succeeds)
```

This says that if the world was your adversary and controlled how long the ODE runs and controls how the slope is changed, could the player succeed in the challenge. Note that the loop iteration is controlled by the player, since choosing to play one more round is strictly the players choice. The reason the ODE is being controlled by the world is because the world is not time-triggered and could choose to change the slope at any point of time. However, the world does need to guarantee that the player would make some progress by moving forward otherwise the player can be defeated by simply not running the dynamics. To avoid that edge case we add a dual test $?\{(t > \epsilon);\}^{@}$ to the world controller.

Given this model of the game, we first prove a simple challenge, the player can walk forward to an arbitrary position along the x-axis.

challenge-move Given an arbitrary position $x_b \geq x$ along the x-axis, we show that it is possible for the character to move past that point. For simplicity, let us assume the character starts with a valid non-zero velocity of 1. Given that we don't know what distance $x_b - x$ is or how long the world will chose to run the ODE, we cannot prove this by unfolding iterations. However, given that the world needs to maintain $t > \epsilon$ we know that in every iteration some progress will be made. Intuitively, this means that in some number of steps the distance will be covered. Ensuring that the x-component of the slopes and the amount of time each iteration is run is atleast $\epsilon > 0$, ensures that the world cannot generate vanishing but non-zero values for these variables. The challenge can be formulated as follows:

```
(x = 0 & xb>=x & v = 1 & constants ) ->
<{
{control player}
{ODEs & h>=l & t<=T}^@
{?(t>eps); control world}^@
}*
> (x > xb)
```

Instead of proving this directly, I show that the following model holds:

```
(x = 0 & xb>=x & v = 1 & constants ) ->
<{
{control player}
{ODEs & h>=l & t<=T}^@
{?(t>eps); control world}^@
}*
> (x > eps^2)
```

That is, the amount of progress that can be performed in each iteration is atleast eps^2, this provides a bound on the number of iterations required to prove $x > xb$. In principle, one could unfold the iterations to prove reachability in the absence of obstacles and avoid Xeno's paradox.

challenge-avoid This challenge models the scenario where there is a point on the ground marked as an obstacle (xb, l) and the goal of the player is to ensure that she does not walk over the obstacle enforcing a jump. The success of this challenge is measured when if $x = xb- > h > l$. Modeling this using loop convergence proved difficult and so I view this as a two part challenge. The first part, uses the previous proof in showing that one can reach a point x on the ground such that $xb <= x + T_j * dx * w$. Now, we know that in a single iteration, if the player jumps, she should be able to reach the target which is the only simplified part the proof models.

```
(x = 0 & xb=x+tj*w & w = 1 & t= T & constants ) ->
<{
{
if(t >= T ){
        t:= 0;
        /*jump if the next iteration is too late,
        this is the worst case scenario (i.e dx=1),
        if you have fallen short, you can jump again. */
        if(x + T >= xb & h=l ){
                j:=J;
                g:=G;
        }
}
}
{ODEs & h>=l & t<=T}^@
{?(t>eps); control world}^@
}*
> (x = xb->h>l)
```

Note that while it may seem that the player can succeed by simply not reaching the target and thereby not needing to be above it, the loop iteration and the initial value for dx is chosen by the world. Since the setup is such that the character can reach xb in one iteration, the world can definitely force it to do so.

This still does not ensure that the iteration cycles would allow the player to jump at the exact moment when composing various challenges and movement cycles. By ensuring that xb is introduced at a distance that covers a few buffer iteration cycles, the design could ensure the playability when composing challenges.

challenge-attack The attack challenge is set up similarly, requiring that when $x = xb- > h = hb$ i.e the character should reach some position $hb > l$. To identify the correct set of starting conditions that would ensure this can be achieved by a jump in the next iteration, we compute the time tb taken to cover the distance between the highest point that the player can reach H_j and hb.

$$\boxed{tb^2 = 2 * (-hj + hb)/G \& tb > 0}$$

If $xb = x + (tb + tj) * w$ and $tb + tj < T$, the player can reach the target in the next iteration. We add a few more simplifications that sets the velocity component arising due to the slope to be 0 while jumping and the linear speed to correspondingly be 1. However, this can be incorporated by assuming (dx, dy) is constant for the iteration. Further, we also simplify the world model to not allow any changes to the slope. This can be incorporated into the overall model by allowing the world to only change when the player is on the ground and not while jumping. The constraint between the jump times and the trigger time is also a reasonable one since it says that jumping iteraions should occur within one trigger cycle. This would allow the character to repeatedly jump after landing, instead of having to wait for the time trigger after each time the character jumps and lands.

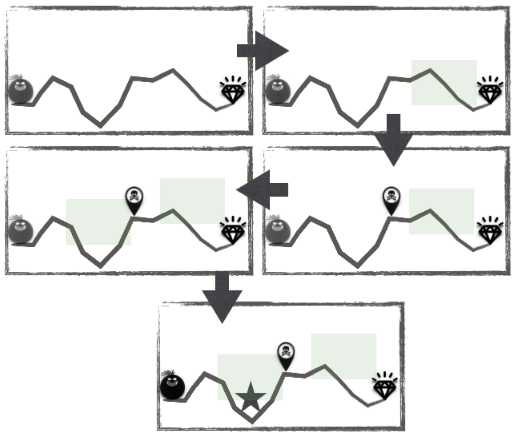
```
(x = 0 & xb=x+tj*w & w = 1 & t= T & constants ) ->
<{
```

```
{
if(t >= T ){
t:= 0;
/*assuming constant linear speed while jumping. */
if(x + (tb+tj)*w >= xb & h=l){
j:=J;
g:=G;
dx:=1;
dy:=0;
}
}
}
{ODEs & h>=l & t<=T}^@
{?(t>eps); control world}^@
}*
> (x = xb->h=hb)
```

The challenge proofs can be shown to hold using the *solve* rule for solving the ODE and performing Quantifier elimination (.kyx and .kya files are included with the submission)

5 Designing a game

Though simple, the individual pieces above cover a significant set of basic features that most physics based platform style or 2D scroller games have. However, to design a complete level one needs to put these steps together. While I could not explored it within the scope of this project, an interactive design system can be envisioned that allows an author to design a game in bottom-up manner, starting from the winning criteria and exploring acceptable regions where game elements can be added. Discontinuities can be bridged by special game elements that allow altering physical realities(like reducing gravity) and player characteristics(increasing jumping speed). A useful feature would be to build a system that allows an interactive way to author diamond proofs and identify constraints.



This illustration shows a possible approach. The designer begins with fixing the end goal of a game (a challenge to reach or attack a jewel). The system then helps identify a buffer zone(green) required for the player to jump in which no other obstacles are placed. The designer can then place a challenge(avoid danger). The system can identify the next buffer zone and indicate if there is a discontinuity (such as jumping speed needs to be modified for the player to achieve this target). The designer can then place a *power-up* element to bridge the gap. Following this, a safe starting position can be computed for the player.

5.1 Playability vs Non-triviality

The focus so far was on ensuring the level is safe and fair to the player and that a reasonable player has some way of achieving the goals set out. It is often the case that a players physical capabilities in the game such as the jumping velocity and maximum height to which the player can reach will not be constant but will be modified based on achieving challenges within the game. Most games even constantly modify gravity to achieve different styles of motion. Once we have a proof that a player can achieve a challenge, one can easily modify game constants such that the proof fails. This is good, since it can be used to prove the non-triviality of a challenge. It can be shown that unless the player achieves a power-up of some nature, the subsequent challenges will be impossible to solve with current capabilities. From the proof, one can identify what rewards a power up should provide in order to pass the next challenge.

6 Discussion

The hope of this project is to tie together various aspects of CPS that was learnt through the semester within the framework of a simple 2D game. As a part of this project I modeled a simple but general level designer that would always allow the player to move and provide sufficient space to jump while keeping the character within the confines of the game world. The player is modeled as a simple point that can move or walk along the designed level. The control allows the player to increase and decrease speed and jump to avoid obstacles or attack them.

Lessons During the initial proposal and white paper, I viewed this project as a hybrid system with opportunities to prove liveness/diamond modalities. While obvious in hindsight, it quickly became apparent that when building a game design, one has to think about an adversarial design world. The world models a general environment and it is unreasonable to expect it to be time-triggered even though sensing and reacting to the world could be time-triggered. For the model, this meant that one had to view the game physics as being driven by the adversarial world. My initial models did not consider dual driven ODEs and in general though the models appear incredibly simple, understanding the adversarial nature of the system was an interesting learning point.

Invariant and variant driven proofs help capture the constraints relevant in making the model safe and playable. These constraints essentially encode design decisions – how close can two obstacles be placed or how high can a platform move. Further, these help in building non-trivial game models by suggesting exactly how far an obstacle needs to be placed such that a gamer needs to perform some action designed for the player. Within game design, discontinuities can be handled by adding specialized game elements such as a power up that bridges the constraints required between two *components* that have been individually verified.

Extensions The game currently models only two explicit actions of the player – walking along the level and jumping. It also models only two challenges or game elements – one that requires the player to avoid a particular region on the level by jumping over it and one that requires the player to reach a particular region by jumping onto it. These two actions as an abstraction cover many game scenarios. However, many more game elements can be introduced to make the design space more rich.

An adversarial enemy attempts to strategically target the player such that she loses the game. This could be a competitive second player or an AI agent. The current system does not model any adversarial enemy. Other adversarial elements in the game such as a target that needs to be attacked or a region that needs to be avoided can be modeled as described but these challenges can be further enriched by allowing these targets to move in complex manners.

Game complexity is often introduced by including resource constraints and opportunities to increase resources within the environment. We briefly talked about time constrained system. Collecting resources can be modeled using target actions. Again, interesting combinations can be considered – If the jumping velocity that the character can avail diminishes over time should she jump quickly to attain a high moving target or wait until the target moves down. A combination of strategies could quickly provide a large number of actions that a character can perform and proving their compositions is a challenging next step.

While player physics can be made arbitrarily complex, the goal of this project was to build upon simple physics and show liveness and safety properties. Incorporating challenging player physics within diamond modalities is another extension that may be of interest.

I also envision an interactive design and proof system that could help visualize a simple model of interest and then allow modification of the model interactively based on proof constraints. This would provide intuition and feedback since the model encodes an inherently spatial design.

References

[1] Adelhardt, Kim, and Nedyalko Kargov. "Mario game solver." IT University of Copenhagen (2012).

[2] Aloupis, Greg, et al. "Classic Nintendo games are (computationally) hard." Theoretical Computer Science 586 (2015): 135-160.

[3] Demaine, Erik D., Giovanni Viglietta, and Aaron Williams. "Super Mario Bros. is harder/easier than we thought." (2016).

[4] Jeannin, Jean-Baptiste, et al. "A Formally Verified Hybrid System for the Next-Generation Airborne Collision Avoidance System (CMU-CS-14-138)." (2014).

[5] Motivating Calculus-Based Kinematics Instruction with Super Mario Bros

[6] http://info.sonicretro.org/SPG:Jumping

[7] Mller, Andreas, et al. "Change and delay contracts for hybrid system component verification." International Conference on Fundamental Approaches to Software Engineering. Springer, Berlin, Heidelberg, 2017.

[8] Icon credits: Joel McKinney, iconsphere, corpus delicti from the Noun project