A Projection-Based Approach for Real-time Assessment and Playability Check for Physics-Based Games

Mohammad Shaker[1], Noor Shaker[2], Mohamed Abou-Zleikha[3] and Julian Togelius[4]

[1]Joseph Fourier University, Grenoble, France
[2]IT University of Copenhagen, Copenhagen, Denmark
[3]Aalborg University, Aalborg, Denmark
[4]New York University, New York City, USA
mohammadshakergtr@gmail.com, nosh@itu.dk, moa@create.aau.dk, julian@togelius.com

Abstract. This paper introduces an authoring tool for physics-based puzzle games that supports game designers through providing visual feedback about the space of interactions. The underlying algorithm accounts for the type and physical properties of the different game components. An area of influence, which identifies the possible space of interaction, is identified for each component. The influence areas of all components in a given design are then merged considering the components' type and the context information. The tool can be used offline where complete designs are analyzed and the final interactive space is projected, and online where edits in the interactive space are projected on the canvas in realtime permitting continuous assistance for game designers and providing informative feedback about playability.

1 Introduction

Game design and content creation is complex and labour-intensive. Therefore, game developers have devoted large efforts to the development of reusable systems to facilitate game design and production. Third-party middleware companies also focus on reusability through providing essential game-independent functionalities such as path finding, animation and physics simulation [1,2]. Most of these tools are designed for experts in game design and development.

Level design is an essential part of the game design process. Level designers put considerable time and effort into creating an interesting, playable level. This process is typically iterative where designers start with a simple sketch, check whether it is playable in its raw form and which parts work and don't work, and go back to the editing mode to change and embellish the level. This process is repeated until the designer is satisfied with the result.

Like for other labour-intensive processes, artificial intelligence techniques could be leveraged to assist and offload the designer. In particular, intelligent authoring tools could ease the design process and decrease time consumption, freeing the designer resource up for higher-level design tasks. Such tools should be easily accessible to game designers and provide comprehensible yet informative feedback [3,4]. There has recently also been much interest in developing tools that automatically check for playability, e.g. through the use of AI agents designed specifically to learn the rules of the game and explore the possible playable space afforded by the design [4].

One notable example of an industry-developed game-specific authoring tool is the one designed for the game *Rayman Legends* [5] which provides a user-friendly interface supported by automatic adjustment of basic features such as lighting, colour, and basic platform. The tool also allows real time editing and adjustment of the different game artefacts.

Recently, there has been increasing interest in academia in the creation of authoring and mixed-initiative tools. Some notable works include *SkectchaWorld* a tool for modelling 3D world through the use of a simple visual interface [6], *Tanagra* a mixed-initiative design tool for 2D platformer in which human and computer collaborate to design game levels [3], *Sentient Sketchbook* a tool that aids the design of game maps and automates playability check [7], *Ropossum* a mixed-initiative design tool that permits automatic generation and testing of physics-based puzzle games [8,9], and the assisted level design tool for the game *Treefrog Treasure* that enables visualisation of the reachability between the game components [10].

One of the core aspects when designing game levels and when implementing a designer-assistance tool is playability. Playability checking could be done in different ways. There are more or less direct approaches, based on e.g. verifying the existence of paths between different points [11], measuring gap widths and drop heights [3] or proving the existence of solutions when the game mechanics or some approximation thereof can be encoded as first-order logic [12]. However, in many cases you need to actually play the level in order to show that it is playable, and thus requires us to build an agent capable of playing the game [13]. Building an agent capable of proficiently playing a given game is rarely straightforward. The results obtained by most agents are usually the path followed to solve the level, if the level is playable [9] or simply a negative feedback indicating an unplayable design. Attention is rarely given to analysing or providing more informative feedback about the full interactive space afforded by the design artefacts. This can be highly beneficial for game designers since it provides information about the unused game components and the playable space in the level.

This paper introduces a new version of an authoring tool for physics-based game, and in particular a fast method for generating feedback to game level designers during the design process. We use the popular physics-based puzzle game *Cut the Rope* as a testbed for our approach. The AI inspects the current state of the game and projects possible future states; in contrast to the method used in a previous version of the tool, which was based on search in a constrained space of game actions, the new method is based on search in a a space of sequences of geometric operations. Part of the result of the method is visualisation of the playable area, i.e. the proportion of the game level in which actual interaction with the player can take place, on the game canvas. The area is plotted on the canvas and updated with every edit by the designer. Thus, the tool permits realtime visual feedback about the interaction space and whether the game is solvable. This is achieved by defining an *influence area* for each game component considering their physical properties and relative position. The final interactive area is the combination of the influence areas of all components presented in a design. Through accounting for the context information and the physical properties, the agent is able to accurately infer the playable space without the need to simulate the game.

The current work builds on previous work by [14] on procedurally generating content for the game using a search-based approach. This work was further extended in [9] by implementing a simulation-based approach for playability testing. The simulation used a Prolog-based agent that plays through the level and considers only the most sensible moves at each state. Although the agent is able to accurately detect a playable design, the method followed suffers from a relatively high processing time (checking a single level takes an average of 29.87 ± 58.28 sec). Furthermore, the agent stops searching after finding the first path that solves the level. Extracting all possible playable trajectories would yield a substantial increase in the processing time.

The purpose of the work presented in this paper is not only to identify the trajectory followed to solve the game, but also to provide an easy to interpret visual illustration of all possible playable paths in real time. We believe that this information is vitally important to game designers who are keen to easily analyse their design and to rapidly realise the impact of their design choices on player experience.

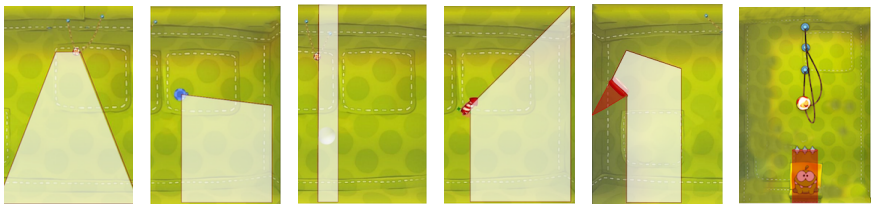
2 Cut the Rope: Play Forever

Cut the Rope is a popular commercial physics-based puzzle video game released in 2010 by ZeptoLab for mobile devices. The game was a huge success and it has been downloaded more than 100 million times. There is no open source code available for the game so we had implemented our own clone using the CRUST engine [15] and the original game art assets. Our clone, *Cut The Rope: Play Forever*, does not implement all features of the original game, but includes all the fundamental features and allows faithful reimplementation of a large portion of the original levels.

3 The Influence Area

The fundamental concept in the method proposed here is the *Influence Area* (IA) Each component in the game has its own physical properties that affect the candy's trajectory, velocity and acceleration. The effect takes place when the candy falls within the IA of that component. A component's IA is is all possible trajectories the candy can follow when interacting with this component. An estimation of the IA is defined differently for each component according to its type as follows:

- Rope: The IA covers the candy's possible trajectories when detached from the rope. Since the physical properties of a rope resemble those of a spring, the rope's IA is defined as a trapezoid where its two legs are defined by the two lines passing through the rope's pin and one of its bases is the horizontal line passing through the candy (see Fig 1(a) for an illustration).
- Air-cushion: Its IA is a trapezoid oriented in its blowing direction: one of its bases is the vertical line passing through the air-cushion, the distance to the other bases is defined based on the air blowing force; one of its legs is defined by a line passing through the air-cushion and angled slightly downwards due to the gravity force while the IA extends to the boundary of the canvas. An illustration is presented in Fig. 1(b).



(a) (b) (c) (d) (e) (f)

Fig. 1. Examples of the basic influence areas for the different game components. Inclusion areas are presented in light colour while exclusion areas are in red.

- Bubble: The basic IA of a bubble is the simplest and is defined as a rectangle surrounding the bubble and extending upwards to the canvas boundaries. This covers the area of the bubble while floating and in the case of being pressed freeing the candy downwards as presented in Fig. 1(c)
- Rocket: Based on the rocket's direction, its IA is a trapezoid covering the candy's possible paths when set free. Consequently, the bases of the trapezoid are the vertical line passing through the rocket and the boundary of the canvas, its legs are the other canvas' boundary and a line starting at the rocket's position and angled according to its direction as can be seen in Fig. 1(d)
- Bumper: The IA of the bumper is the hardest to define due to its complex physical property and the dependency between its IA and the direction of the collision with the candy. The basic IA is defined as a six-point polygon covering all possible bouncing trajectories after the collision considering the effect of gravity. An example can be seen in Fig. 1(e).

The interactive space in a given design can then be identifying by combining the IAs of all components presented. We call this the *Projection Area* (PA). Note that the previously defined areas are carefully chosen approximations of the exact areas which could be more accurately determined by applying the laws of physics. We decided to consider these approximations instead of the exact calculations since they yield almost equally accurate results while being considerable computationally cheaper.

While the previously defined IAs are relatively easy to calculate, these are the most basic form of IAs. For complicated designs with more than one component, the IA of each component will typically be altered by the presence of another. Things get more complicated as the number of components increases and a more advanced solution becomes a necessity. The solution we propose differentiates between two types effects according to how the components' IAs interact: components of *inclusive* and *exclusive* effects. Components with inclusive effects are characterised by IAs that increases the PA already established either by expanding or by adding a new separated range. Exclusive effects, on the other hand, decrease the size of the PA through subtracting parts that became inaccessible as a result of adding a component with such property. Some of the components exhibit properties from the two types depending on their type and the other component of interaction. This applies for example to bumpers which expand the area

in the direction that faces the candy and exclude the one in the other direction. Fig. 1 presents some examples for both cases.

Given the components' characteristics and IAs, an agent can be implemented to provide an online visual feedback and playability check of complete designs or while the level is being designed through automatically expanding and/or reducing the final playable area as new components are being added or removed.

In the following sections, we describe the steps followed to identify the projection area using a projection agent.

4 The Projection Algorithm

The heart of the proposed method is the projection algorithm. This algorithm handles the interactions between the components and effectively and accurately defines the interactive area. The basic idea is that by starting with the ropes to which the candy is attached, and recursively expanding and/or subtracting fragments of the PA, we can ultimately deduce the area that is reachable by the candy. Eventually, we can assess whether a level is playable by examining the overlap between the resulting PA and Om Nom's position. If such an overlap exists, the level is playable. We considered a number of implementation choices when constructing the agent. In order to support these choices and clarify the reasoning behind them, we will start by explaining two scenarios where a naive implementation of the basic idea will not work.

Scenario 1: Order effect Combining the IAs of all components in one pass following Equation 1 to build the final PA is not a feasible solution. This is mainly due to the nature of the game where the IA of a component depends primarily on the candy's current position, direction, and velocity and the order in which components are activated.

$$PA = \bigcup_{i=1}^{Components} IA_i \tag{1}$$

Consider an interaction between two ropes, two bumpers, two rockets and the candy as an example (Fig. 2). Applying the IAs of the components in a different order will lead to different results. For instance, if we apply the IAs in the order: ropes, rockets and finally bumpers, the result will be the PA presented in Fig. 2(a). As can be seen in this case, the IAs of the bumpers block the candy from reaching one of the rockets that lead to Om Nom and as a result, the level will be classified as unplayable. On the other hand, if we follow another order so that the IAs of the rockets are applied last, the resultant area will cover Om Nom and the level will be playable (Fig. 2(b)).

The order in which the components are handled is therefore essential since different order will lead to different PAs and consequently different judgement about playability. While it is vitally important to determine the sequence in which the components should be executed, this order is not obvious.

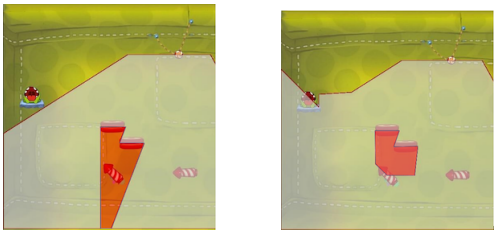


Fig. 2. Two different results of applying the IAs in different orders. The first image shows the IAs when applying: ropes, rockets and finally bumpers in order. The second image shows the IAs when applying: ropes, bumpers then rockets.

Scenario 2: Solution path One possible solution to the problem described in the previous section is to process the components according to the possible paths that could be followed to solve the game. This approach will also fail since it is highly likely that there will be more than one component that could be activated at any specific time during the game. The right order in which these components are handled can not be easily accurately defined (we will end up facing the same issue discussed in the first scenario). Furthermore, our goal is to define an area that constitutes all reachable positions and not only the ones that lead to the solution and this can not be achieved by relying solely on the solution path.

4.1 The Projection Algorithm

The above mentioned scenarios emphasise the need to a more intelligent strategy that considers rich context information and handles the ordering effect. This can be achieved by constructing an agent that processes the components recursively one at time, analysing the game state and considering the possible interactions with other components in a depth-first approach until a stopping criteria is satisfied. The algorithm proposed is a search method that exhaustively traverses the full tree of possible paths. The resultant PA is the combination of all PAs for all paths explored. This permits extraction of the the full playable area (those interaction areas that lead to a solution and those that do not). We intentionally do not discard unsolvable paths since our goal is to help designers visualise the area that is used and in which players can take actions regardless of their outcome. The algorithm followed (Fig. 1) can be described as follows:

The algorithm handles two main data structures: a sorted list of components, *coveredComps*, and a PA (polygons represented as a list of points) that is constantly updated as new paths are explored. The algorithm starts by calculating the IAs of the ropes since the game always begins by cutting them to initiate the candy's movement. The components covered by the ropes are calculated and added to a set of covered components. The algorithm then iterates until the tree representing all possible paths is explored. In each iteration, a set containing the components covered by the IA of the currently processed component is calculated (line 2). The algorithm then checks if any of the covered components can be excluded (line 3) and removed from the set. A recursive call is then

Algorithm 1: The Projection Algorithm

```
   Data: List of components with their initial positions;
   PA = ropes.applyIA();
   coveredComps = PA.findCovered(allComps);
1  while (not(empty(coveredComps)) do
2      coveredComps = PA.findCovered(allComps);
3      coveredComps = removeExcluded(coveredComps);
4      for cc in coveredComps do
5          PA += cc.applyIA();
6          projectionAlgorithm(PA);
```

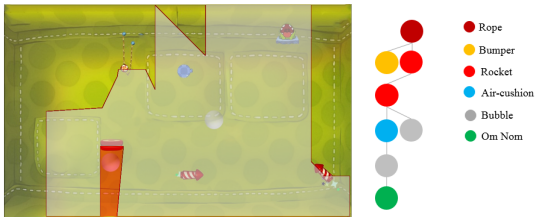


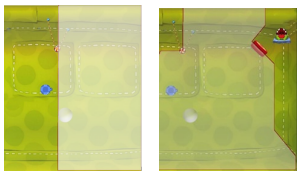
Fig. 3. Example level and the corresponding explored tree.

performed; the IA for each item in the set is calculated (line 5) followed by a recursive call to the algorithm with the newly formed PA (line 6).

Note that although the algorithm performs exhaustive search, the method takes on average 0.1 ± 0.02 sec (over 100 runs) to process complete designs of an average of 8.53 ± 1.74 components. This is because only few components become cover in every iterations and each level usually contains a small number of components. Fig. 3 presents an example level and the full tree explored to calculate the PA. A step-by-step example of how the PA is calculated is presented in Section 6.1, however this example is presented to draw attention to the size of the tree and to clarify the very short amount of processing time required to traverse it.

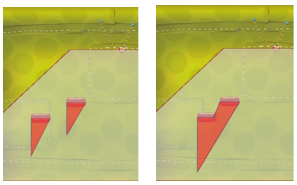
The success of the projection algorithm depends on the following important details:

Estimating the Interaction Direction An important factor that defines the IA of a component is the direction of interaction with the candy. For example, the IA of a bumper is reversible according to the direction of the collision. Simulating the game is one possible way to precisely set the direction. However this solution is too slow in many cases (an average of 29.87 ± 58.28 sec is required to simulate one level [9] and the simulation has to be repeated for every edit). Moreover, we are not interested in the exact position but with a reliable estimation of the direction. For these reasons, the candy's direction is calculated according to the position of the last component with which the candy interacted. The algorithm proposed solves this issue by considering the order in which they become covered. Specifically, the direction of interaction with a component n is the position of the component $n - 1$ in the recursive call.



(a) (b)

Fig. 4. The extended influence area of a bubble being covered by an air-cushion (a), and a bumper being covered by a bubble (b).



(a) (b)

Fig. 5. Examples of the IA of two bumpers according to the distance between them. In (a), the bumpers are processed separately while in (b) their exclusion areas are combined since they are closer.

Handling Context We previously discussed the basic IA of each component and mentioned that the IA of one component is usually affected by the context. In the following, the IAs of some components are refined to account for their interaction (the IAs of the other components remain intact):

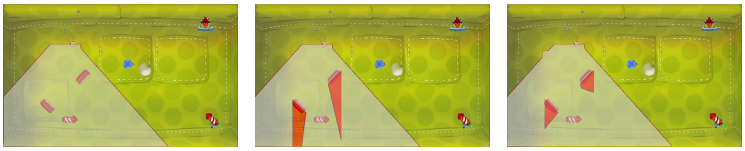
- Bubble: The presence of other components such as an air-cushion, a bumper or a rocket changes the IA of a bubble. When covered by a an air-cushion for example, the IA of the bubble extends so that the new area is a wider rectangle due to the effect of the air force.
- Bumper: The IA of a bumper is altered relative to the type of the other components placed nearby: the existence of a nearby bubble and an air-cushion, for instance, results in a new extended area as can be seen in Fig. 4. Also, the presence of another nearby bumper forms a new combined exclusive area as can be seen in Fig. 5.

When accounting for the above factors, the projection algorithm is capable of effectively constructing the PA and successfully detecting when the design is playable.

4.2 Implementation Details

The Clipper library [16] is used to preform polygon clipping. We used the C# version of the library and interface it with our physics engine (CRUST) to simulate the game[1].

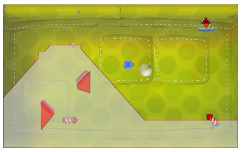
[1] A video showing the method working is available online: http://noorshaker.com/CutTheRope.html



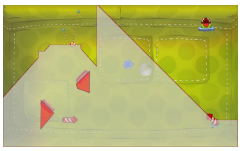
(a) Applying the ropes' IA

(b) Calculating the bumpers' exclusion area

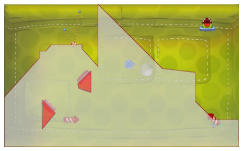
(c) Applying the bumpers' inclusive area



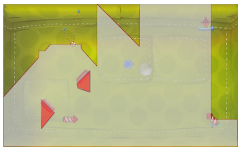
(d) Adding the rocket's IA



(e) Applying the rocket's IA



(f) Extending the IA according to the air-cushion



(g) Adding the IA of the bubble

Fig. 6. The offline use of the algorithm. Projection area is presented in a light colour and excluded areas are presented in red.

5 Online and Offline Assessment

The projection algorithm presented and the scenarios discussed assume that the components and their positions are known and provided as input to the algorithm. In this case, the algorithm is used as a tool for post-design assessment of a level where the final interaction area is projected to give the designer a visual feedback. This information can be used to alter the initial design and changes to the PA can be visualised accordingly either in the offline mode (after all adjustments are made) or through the online mode.

In the online mode, the projection algorithm is activated while the scene is being edited. In this case, whenever a new component is placed in the canvas, the algorithm is called with the list of all components used so far. If the newly added component has an exclusive property, the PA is recalculated through a new run of the algorithm. Otherwise, the PA is expanded according to the type of the recently added component. This might result in covering other components in the scene and therefore the algorithm is called with the set of covered components only as input.

The IAs of the components were carefully tuned and the method has been tested on several cases and showed promising results in terms of accurately estimating the PA and identifying solvable designs. There is however no guarantee that there are no levels on which the algorithm fails ((un)fortunately, we cannot report any). If such situations exist, we do not know whether the algorithm is biased towards false positives (cases where the method classifies unsolvable levels as solvable) or false negatives.

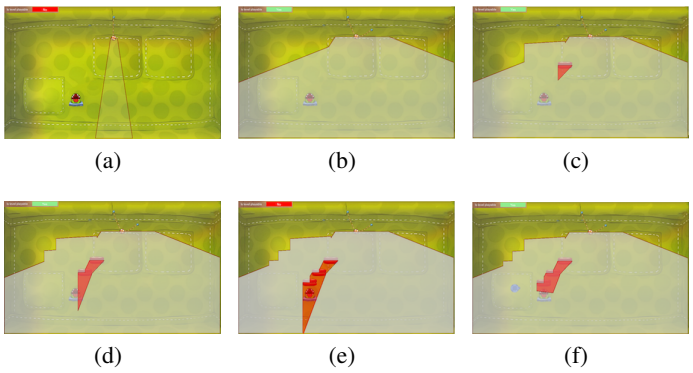


Fig. 7. The online use of the algorithm in Ropossum authoring tool. Projection area is presented in a light colour and excluded areas are presented in red.

6 Showcases

Below we present two examples that showcase the algorithm's capabilities during both the offline and the online mode. Note that the algorithm works exactly the same in both cases and we differentiate between them to clarify and emphasise its capabilities.

6.1 Showcase 1: Offline Mode

In this example, a complete design is provided as input to the system to calculate and visualise the resulting playable area. Fig. 6 presents an illustration of the steps followed by the algorithm. We start by calculating the ropes' IAs which in this case cover two bumpers and a rocket (Fig. 6(a)). Since the algorithm sorts the components so that bumpers are handled first, the areas excluded by the bumbers according to the candy's current position are then subtracted from the PA (Fig. 6(b)). Their coverage areas are then added to the PA, reincluding fragments of the area previously excluded (Fig. 6(c)). Continuing to handle the rest of the component, the IA of the covered rocket is then applied extending the PA to cover the other rocket (Fig.6(d)). The IA of the newly added rocket covers the air-cushion and the bubble (Fig. 6(e)). Recursively handling these two components results in first adding the IA of the air-cushion (Fig. 6(f)) then expanding the resultant area according to the IA of the bubble (Fig. 6(g)). At this stage, and since there are no more components to handle, the process is terminated and the level is considered playable. The visualisation of the PA in this case clearly illustrates that all components are accessible starting from the candy's initial position. It also reveals the active space of the canvas where actual interaction can take place.

6.2 Showcase2: Online Mode

The online use of the algorithm is illustrated in Fig. 7. We start by a simple level design of one rope and Om Nom (Fig. 7(a)). The player/designer can add, remove and/or

Table 1. Comparison between the current approach and the previous ones. The table presents the time required to check if a design is playable (in seconds), the time required to evolve a playable design (in seconds) and the number of generations to reach the first valid solution.

Agent	$Playability$	$Evolution_{time}$	$Generations$
Pseudo-Random [14]	0.98 ± 0.64	21.27 ± 23.44	2.63 ± 1.82
Simulation-Based [9]	29.87 ± 58.28	470.1 ± 525.4	2.48 ± 1.58
Projection-Based	0.1 ± 0.02	-	-

reposition components using an authoring tool [8]. A green flag (top left) is used to indicate a playable design when adding a second rope (as in Fig. 7(b)). During the design process, the PA is continuously updated as the game state changes. Fig. 7(c), 7(d) and 7(e) presents the updated PA after positioning three bumpers leading to the exclusion of Om Nom and as a consequence a detection of an unplayable level. This situation is overcame when the designer adds an air-cushion (Fig. 7(f)) which resets the flag.

7 Ropossum Integration and Performance

The projection algorithm is incorporated as part of the authoring tool *Ropossum* [8]. Some of the functionalities previously provided are the automatic generation of playable content [14] and the automatic check for playability [9]. With projection areas as a newly added feature, designers can benefit from editing procedurally generated levels and visualising PAs of complete designs or during the level generation process. The method also supports playability check which can be done faster than the previous methods as presented in Table 1.

8 Conclusions

This paper presents a method for visualising and checking playability for physics-based puzzle games. The method proposed considers the physical properties of different game components and accordingly defines their space of influence. Informative combination of the influence areas of all components in a given design defines the playable space in which interaction with the player can take place. This was effectively achieved through a careful consideration of the context information. The algorithm executes in a short time and provides informative visual feedback both in the offline mode when the agent is set to work on a complete design and in the online mode while the level is being designed. While the method differs considerably from all previous methods for game level analysis that we know of in that it is based on the novel concept of Projection Areas and search in projection space, it is still at its core an artificial intelligence method that performs search in a problem- and domain-appropriate space.

There are a number of physics-based puzzle games where we see the proposed approach easily applicable. We are thinking of games such as Amazing Alex, Sprinkle, iBlast, Moki, Enigmo 2,Touch Physics 2, Chalk Ball, and Angry Birds; these games have similar properties to Cut the Rope in terms of in-game entities and their movement

and interaction. Generalising the method to work for any game of this genre, perhaps by encoding mechanics in a description language for physical puzzle games, would be exciting future work.

Acknowledgments

We thank ZeptoLab for giving us permission to use the original Cut The Rope graphical assets for research purposes. The research was supported in part by the Danish Research Agency, Ministry of Science, Technology and Innovation; project "PlayGALe" (1337-00172).

References

1. Havok, 2011. Havok, www.havok.com.
2. Interactive Data, 2011. SpeedTree.
3. G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.
4. Adam M Smith. Open problem: Reusable gameplay trace samplers. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2013.
5. Ubisoft, Ubisoft Montpellier, and Digital Eclipse, 1995. Rayman , Ubisoft.
6. Ruben Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. Integrating procedural generation and manual editing of virtual worlds. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, page 2. ACM, 2010.
7. Antonios Liapis, Georgios Yannakakis, and Julian Togelius. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of ACM Conference on Foundations of Digital Games*, 2013.
8. Mohammad Shaker, Noor Shaker, and Julian Togelius. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press*, 2013.
9. Mohammad Shaker, Noor Shaker, and Julian Togelius. Evolving playable content for cut the rope through a simulation-based approach. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2013.
10. Aaron Bauer, Seth Cooper, and Zoran Popovic. Automated redesign of local playspace properties, 2013.
11. J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G.N. Yannakakis. Multi-objective exploration of the starcraft map space. In *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pages 265–272. Citeseer, 2010.
12. Adam M Smith and Michael Mateas. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):187–200, 2011.
13. J. Ortega, N. Shaker, J. Togelius, and G.N. Yannakakis. Imitating human playing styles in super mario bros. *Entertainment Computing*, 2013.
14. Mohammad Shaker, Mhd Hasan Sarhan, Ola Al Naameh, Noor Shaker, and Julian Togelius. Automatic generation and analysis of physics-based puzzle games. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.
15. Ian Millington. *Game physics engine development*. Morgan Kaufmann Pub, 2007.
16. A. Johnson. Clipper - an open source freeware library for clipping and offsetting lines and polygons, 2014. http://www.angusj.com/delphi/clipper.php.