The Q*bird Level Designer: User-assisted Procedural Level Design in Augmented Reality

Yi Fei Cheng Aline Normoyle

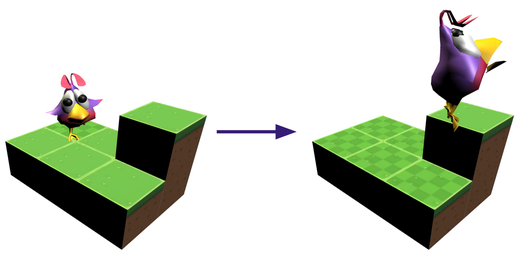
Department of Computer Science, Swarthmore College

ABSTRACT

Augmented reality (AR) gaming is becoming widely available thanks to improvements in hand-held devices such as phones and tablets. In this work, we describe our system for generating levels for the AR game, **Q*bird**.

In **Q*bird**, the player must visit every cell in the level while avoiding bees and cannon balls, similarly to the 1982 arcade game, Q*bert. To create a new level, designers place game elements using virtual cards. The system then generates the remainder of the level, ensuring that it's navigable. Designers can edit these levels by dragging and dropping the created geometry. To test, the designer can drop a character into the level and play it. This system aids playtesting and level design by allowing levels to be quickly specified and tested in the same environment in which the game is played. Furthermore, this system offers an example of how the design of AR levels can also be performed in AR.

Q*BIRD: THE GAME

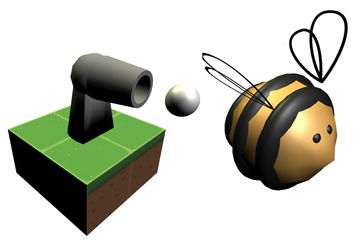


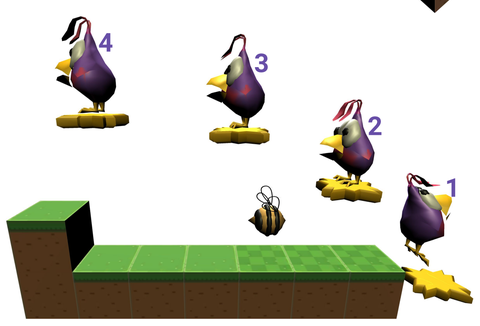
GAME OBJECTIVE

The player must visit every block in the terrain while avoiding enemies.

Q*BERT-INSPIRED ENEMIES

The bee and cannon characters can push the player off the platform.





ESCAPING FROM ENEMIES

To help players escape enemies when cornered, single-use hover disks can be used to teleport to safety.

Q*BIRD: THE LEVEL DESIGNER

The main goal of the level designer is to help a user search the large space of possible levels given a placement of cards.

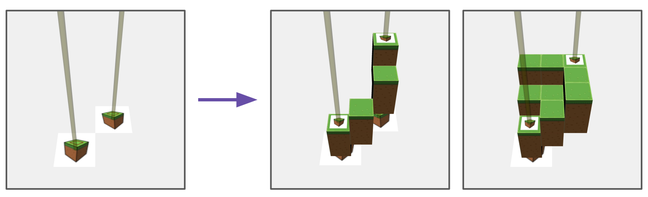
PLACE CARDS

The user can specify the location of level components and waypoints using virtual cards. To create a new level, designers can drag and drop cards displayed below onto a surface. A level can then be generated based on the card configuration.

BEE HOVER DISK CANNON WAYPOINT

EXPLORE

The user can quickly iterate through a series of visually distinct levels that are procedurally generated based on the card configuration.

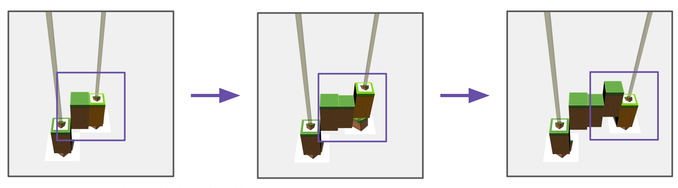


INPUT: An arbitrary configuration of level generation cards

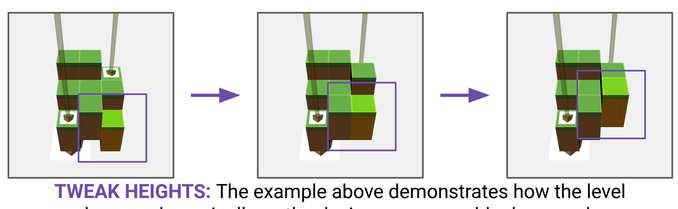
OUTPUT: Two example procedurally generated levels

TWEAK

The user can fine-tune levels by adjusting individual block heights and virtual card positions.



TWEAK CARD POSITIONS: The example above demonstrates how the level changes dynamically as the designer moves a card to the right.



TWEAK HEIGHTS: The example above demonstrates how the level changes dynamically as the designer moves a block upwards.

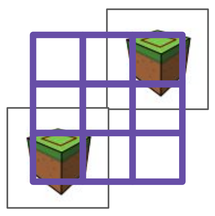
METHOD: THE LEVEL GENERATION ALGORITHM

The system maintains two representations of the level:

- A genome representation: A vector of integers which controls how the level is generated
  - The integer vector consists of two parts: a **PATH GENOME** (determines the order in which neighbors are explored), a **HEIGHT GENOME** (determines the relative heights between neighboring cells, either -1, 0, or 1)
- A phenotype representation: A 2D array of heights which represents the geometry

2D GRID CALCULATION:

We fit a bounding box around the cards placed by the user. The bounding box is then subdivided into block-sized cells to produce a 2D grid encompassing the cards. Ultimately, we represent platform levels as undirected graphs superimposed on this 2D grid of cells. Each node of the graph will have an associated height with it and will map to a cell (row, column) in the 2D grid space.



A 2D grid mapped onto a sample set of waypoint cards.

ROLE OF THE LEVEL GENERATION CARDS:

Cards constrain the level generation in two ways:

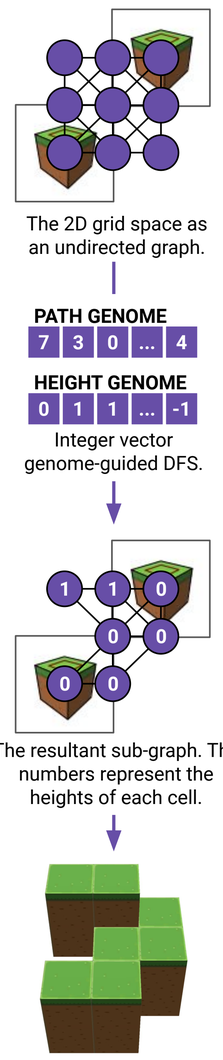
- Cards can define **WAYPOINTS**: Guaranteed level platform positions (the bee and waypoint cards)
- Cards can define **OBSTACLES**: Positions the level platform cannot occupy (the hover disk and cannon cards)

The corresponding game element will be created in the cell the card occupies.

FROM GENOME TO GEOMETRY:

We compute the graph from our cards and integer vector using an algorithm based on depth-first-search (DFS).

We first define an undirected graph using the 2D grid space. We select a cell containing a waypoint card as the starting node. We then carve out a subgraph by traversing between waypoints using DFS, visiting nodes in the order specified by the **PATH GENOME**. Afterwards, we traverse the subgraph using DFS again, setting the height of each subsequent node using the **PATH GENOME**. Lastly, We iterate through each node in the subgraph to place platform level blocks on top of the play surface, producing the geometry.



ALGORITHM PSEUDOCODE:

```
procedure GenerateLevel (g,C)
inputs: Genome g, List of cards C
outputs: UndirectedGraph G
G = InitializeGraph(C);
W = ExtractWaypoints(C);
for i = 0; i < Length(W) - 1; i = i + 1 do
    G = DFS_SetPath(g, C[i], C[i + 1], G);
end
G = DFS_SetHeights(g, G);
return G;
```

METHOD: EVOLUTIONARY SEARCH

We use evolutionary search to ensure the system shows visually distinct examples (e.g. maximize difference) in **EXPLORE** and to minimize the difference between the shared regions in **TWEAK** (e.g. minimize difference when moving cards).

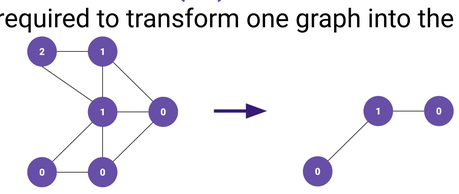
In our case, the genotype is our integer vector and the phenotype is the output platform level geometry. We introduce mutations to the genotype by changing the visiting order encoded in the **PATH GENOME** or editing the relative heights encoded by the **HEIGHT GENOME**. The equations representing quantifying visual difference for the **EXPLORE** and **TWEAK** operations are as follows:

$$\lambda_{explore} = \max_{p \in P} \sum_{i}^{N} (w_1 GE(L_i, p) + w_2 H(L_i, p) + w_3 E(L_i, p))$$

$$\lambda_{tweak} = \min_{p \in P} w_1 GE(L, p) + w_2 H(L, p)$$

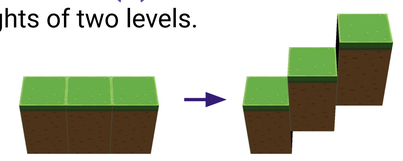
- w_1, w_2, and w_3 are weights determined via trial and error.
- We quantify visual difference with the functions *GE*, *H*, and *E*
- *GE* measures the graph edit distance between a previous level L_i (or L) and the mutated level p.
- *H* measures the pairwise difference in height.
- *E* measures the difference in percentage of empty cells.
- For **EXPLORE** we compare p with N previous levels, while for **TWEAK** we only compare p to one previous level L.

GRAPH EDIT DISTANCE (*GE*): We measure the number of operations required to transform one graph into the other.



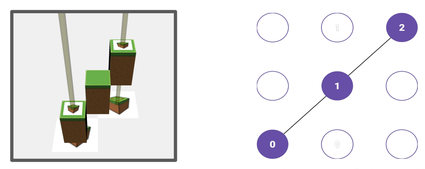
In the example above, we have to perform seven edge deletions and three vertex deletions. Therefore, the GED between the two graphs is ten.

HEIGHT DIFFERENCE (*H*): We measure the element-wise difference heights of two levels.



The height change score between the two example levels presented above is three.

DIFFERENCE IN PERCENTAGE OF EMPTY SPACE (*E*): We find the difference in the percentage of empty cells between two levels.

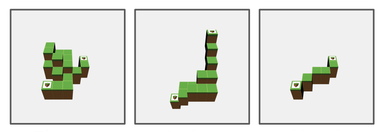


Assuming we map to a 3-by-3 2D grid, approximately 66% of the cells are empty in the example above.

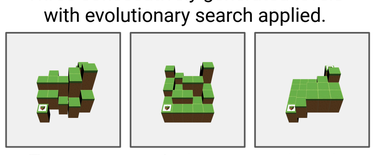
RESULTS

EVOLUTIONARY SEARCH:

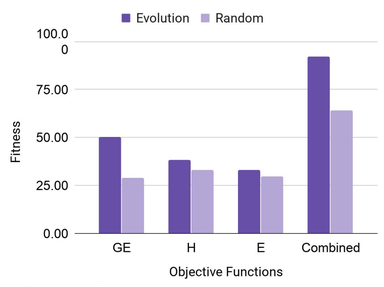
Generating levels randomly can result in many similar-looking levels. We are able to guarantee a higher degree of visual difference between levels in **EXPLORE** by applying evolutionary search.



Three consecutively generated levels with evolutionary search applied.



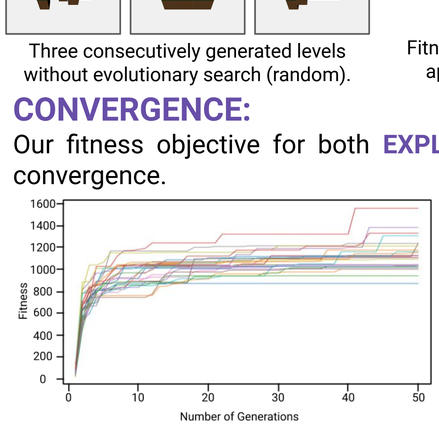
Three consecutively generated levels without evolutionary search (random).

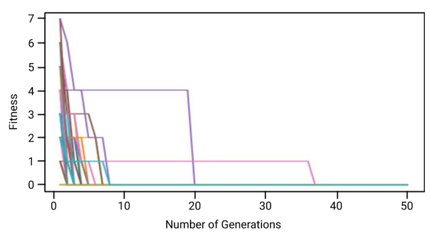


Fitness for when evolutionary search is applied versus random generation.

CONVERGENCE:

Our fitness objective for both **EXPLORE** and **TWEAK** achieve convergence.



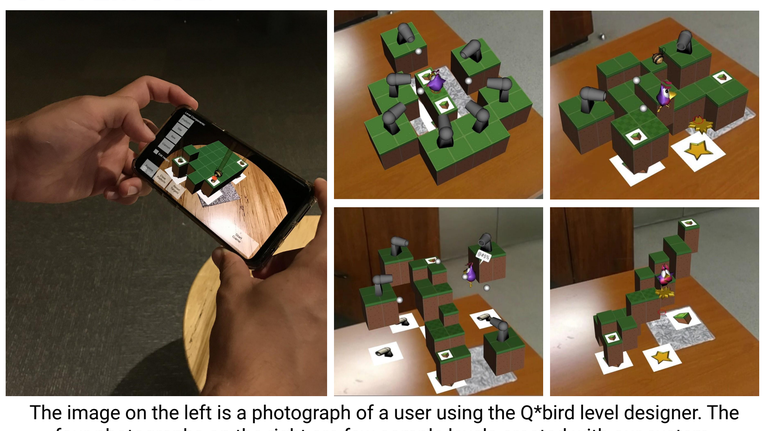


Here, we show the convergence of the fitness objective for **EXPLORE** (top) and **TWEAK** (bottom). In both, we generate 50 random levels of 5-by-5 cells, initialized using 2 platform cards. For **EXPLORE**, our system maximizes visual novelty. For **TWEAK**, we allow our users to make edits to the card positions while minimizing the difference between the remaining parts of the level. Each line represents a randomly picked example. The fitness (Y-axis) of the population improves with the number of generations (X-axis). When exploring, we cap the search generations at 10 and for moving cards, we cap the search generation at 15.

RUNTIME:

- **EXPLORE:** 1s for a 5-by-5 grid, 4s for 10-by-10
- **TWEAK CARD POSITION:** 1s for 5-by-5, 5s for 10-by-10
- **TWEAK HEIGHT:** 0.2s for 5-by-5, 1.5s for 10-by-10

SYSTEM USE:



The image on the left is a photograph of a user using the Q*bird level designer. The four photographs on the right are few sample levels created with our system.

DISCUSSION & FUTURE WORK

In this work, we created a system which combines direct editing and evolutionary search. The resulting levels are organic and visually appealing.

RESULTS FROM PRELIMINARY USER TESTING:

- Drag and drop of cards is intuitive
- AR controls initially unintuitive for some users who expected touch-based mobile phone controls

FUTURE WORK:

- Study of character controllers in AR
- Study how level design affects game aesthetics, player frustration, and player accomplishment
- Collaborative game design